Package: choicer (via r-universe)

October 22, 2026

Title Discrete Choice Models for Economic Applications

Version 0.2.1

Description Fast estimation of discrete-choice models for applied economics. Frequentist likelihoods, analytical gradients, and Hessians are implemented in C++ with 'OpenMP' parallelism, scaling efficiently to specifications with many alternative-specific constants. Compiled Gibbs samplers provide Bayesian multinomial probit and hierarchical models. Post-estimation routines cover predicted shares, own- and cross-price elasticities, diversion ratios, willingness to pay, and welfare counterfactuals. Supports multinomial logit ('MNL'), mixed logit ('MXL'), nested logit ('NL'), Bayesian multinomial probit ('MNP'), and hierarchical Bayesian multinomial logit and probit ('HMNL', 'HMNP').

License LGPL (>= 3)

URL https://github.com/fpcordeiro/choicer, https://fpcordeiro.github.io/choicer/

BugReports https://github.com/fpcordeiro/choicer/issues

Encoding UTF-8

Depends R (>= 4.1.0)

LinkingTo Rcpp, RcppArmadillo

Imports data.table, graphics, nloptr, randtoolbox, Rcpp, stats, utils

Suggests testthat (>= 3.0.0), numDeriv, future.apply, goftest, knitr, rmarkdown

VignetteBuilder knitr

LazyData true

Config/Needs/website pkgdown

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Fernando Cordeiro [aut, cre, cph]

Maintainer Fernando Cordeiro <fernandolpcordeiro@gmail.com>

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

Date/Publication 2026-09-04 21:20:02 UTC

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

RemoteRef HEAD

RemoteSha e8ee4354a426f0eab2a688a80f82330a522ff5f5

Contents

blp . . . 4
blp.choicer_mnl . . . 4
blp.choicer_mxl . . . 5
blp.choicer_nl . . . 7
blp_contraction . . . 8
coef.choicer_fit . . . 9
coef.choicer_hb . . . 10
coef.choicer_mnp . . . 10
consumer_surplus.choicer_hmnl . . . 11
diversion_ratios.choicer_hb . . . 14
diversion_ratios.choicer_mnl . . . 15
diversion_ratios.choicer_mxl . . . 16
diversion_ratios.choicer_nl . . . 17
elasticities.choicer_hb . . . 18
elasticities.choicer_mnl . . . 19
elasticities.choicer_mxl . . . 20
elasticities.choicer_nl . . . 21
ess . . . 22
get_halton_normals . . . 22
gof . . . 23
logLik.choicer_fit . . . 24
logsum.choicer_hmnl . . . 25
mc_asymptotics . . . 27
mcse . . . 28
mode_choice . . . 29
monte_carlo . . . 31
mxl_blp_contraction . . . 32
new_choicer_sim . . . 34
nl_blp_contraction . . . 34
nobs.choicer_fit . . . 36
nobs.choicer_hb . . . 37
nobs.choicer_mnp . . . 37
ppc_shares . . . 38
predict.choicer_hb . . . 39
predict.choicer_mnl . . . 40
predict.choicer_mxl . . . 41

predict.choicer_nl . . . 43
prepare_hmnl_data . . . 44
prepare_hmnp_data . . . 47
prepare_mnl_data . . . 49
prepare_mnp_data . . . 50
prepare_mxl_data . . . 52
prepare_nl_data . . . 54
print.choicer_cs . . . 55
print.choicer_fit . . . 56
print.choicer_gof . . . 56
print.choicer_hb . . . 57
print.choicer_mnp . . . 58
print.choicer_wtp . . . 58
print.summary.choicer_hb . . . 59
print.summary.choicer_mnl . . . 59
print.summary.choicer_mnp . . . 60
print.summary.choicer_mxl . . . 61
print.summary.choicer_nl . . . 62
recovery_table . . . 63
rhat . . . 64
run_hmnlogit . . . 65
run_hmnprobit . . . 68
run_mnlogit . . . 70
run_mnprobit . . . 73
run_mxlogit . . . 75
run_nestlogit . . . 79
sample_by_choice . . . 81
set_num_threads . . . 83
simulate_hmnl_data . . . 83
simulate_hmnp_data . . . 85
simulate_mnl_data . . . 86
simulate_mnp_data . . . 87
simulate_mxl_data . . . 89
simulate_nl_data . . . 90
summary.choicer_hb . . . 91
summary.choicer_mnl . . . 92
summary.choicer_mnp . . . 93
summary.choicer_mxl . . . 94
summary.choicer_nl . . . 95
thread_info . . . 96
traceplot . . . 96
traceplot.choicer_hb . . . 97
vcov.choicer_fit . . . 98
vcov.choicer_hb . . . 99
vcov.choicer_mnp . . . 100
wesml_vcov . . . 101
wesml_weights . . . 102
wtp.choicer_hb . . . 104

Index **106**

`blp` *BLP contraction mapping*

Description

Finds the ASC (delta) parameters such that predicted market shares match target shares, using the contraction mapping of Berry, Levinsohn, and Pakes (1995) doi:10.2307/2171802.

Usage

```
blp(object, target_shares, ...)
```

Arguments

`object`	A fitted model object.
`target_shares`	Numeric vector of target market shares (length J).
`...`	Additional arguments passed to methods.

Value

Converged delta (ASC) vector.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
blp(fit, target_shares = rep(1/J, J))
```

`blp.choicer_mnl` *BLP contraction mapping for multinomial logit model*

Description

BLP contraction mapping for multinomial logit model

Usage

```
## S3 method for class 'choicer_mnl'
blp(
  object,
  target_shares,
  delta_init = NULL,
  tol = 1e-08,
  max_iter = 1000,
  ...
)
```

Arguments

object	A choicer_mnl object fitted with keep_data = TRUE.
target_shares	Numeric vector of target market shares. Length J_inside when no outside option, or J_inside + 1 (with the outside option's share at index 1) when include_outside_option = TRUE.
delta_init	Initial guess for delta (ASC) values. If NULL, uses the estimated ASCs from the fitted model.
tol	Convergence tolerance (default 1e-8).
max_iter	Maximum iterations (default 1000).
...	Additional arguments (ignored).

Value

Converged delta (ASC) vector.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
blp(fit, target_shares = rep(1/J, J))
```

---

blp.choicer_mxl *BLP contraction mapping for mixed logit model*

Description

BLP contraction mapping for mixed logit model

Usage

```
## S3 method for class 'choicer_mxl'
blp(
  object,
  target_shares,
  delta_init = NULL,
  tol = 1e-08,
  max_iter = 1000,
  ...
)
```

Arguments

object	A choicer_mxl object fitted with keep_data = TRUE.
target_shares	Numeric vector of target market shares. Length J_inside when no outside option, or J_inside + 1 (with the outside option's share at index 1) when include_outside_option = TRUE.
delta_init	Initial guess for delta (ASC) values. If NULL, uses the estimated ASCs from the fitted model.
tol	Convergence tolerance (default 1e-8).
max_iter	Maximum iterations (default 1000).
...	Additional arguments (ignored).

Value

Converged delta (ASC) vector.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), w1 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mxlogit(
  data = dt, id_col = "id", alt_col = "alt", choice_col = "choice",
  covariate_cols = "x1", random_var_cols = "w1", S = 50L
)
blp(fit, target_shares = rep(1/J, J))
```

`blp.choicer_nl` *BLP contraction mapping for nested logit model*

Description

BLP contraction mapping for nested logit model

Usage

```
## S3 method for class 'choicer_nl'
blp(
  object,
  target_shares,
  delta_init = NULL,
  damping = 1,
  tol = 1e-08,
  max_iter = 1000,
  ...
)
```

Arguments

`object`	A `choicer_nl` object fitted with `keep_data = TRUE`.
`target_shares`	Numeric vector of target market shares. Length `J_inside` when no outside option, or `J_inside + 1` (with the outside option's share at index 1) when `include_outside_option = TRUE`.
`delta_init`	Initial guess for delta (ASC) values. If `NULL`, uses the estimated ASCs from the fitted model.
`damping`	Contraction damping factor in (0, 1] (default 1).
`tol`	Convergence tolerance (default 1e-8).
`max_iter`	Maximum iterations (default 1000).
`...`	Additional arguments (ignored).

Value

Converged delta (ASC) vector.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 4
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, nest := rep(c(1L, 1L, 2L, 2L), N)]
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
```

```
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_nestlogit(dt, "id", "alt", "choice", c("x1", "x2"), "nest")
blp(fit, target_shares = rep(1/J, J))
```

`blp_contraction` *BLP95 contraction mapping to find delta given target shares*

Description

BLP95 contraction mapping to find delta given target shares

Usage

```
blp_contraction(
  delta,
  target_shares,
  X,
  beta,
  alt_idx,
  M,
  weights,
  include_outside_option = FALSE,
  tol = 1e-08,
  max_iter = 1000L
)
```

Arguments

`delta`	J x 1 vector with initial guess for deltas (ASCs)
`target_shares`	J x 1 vector with target shares for each alternative
`X`	sum(M) x K design matrix with covariates. M[i] x K matrix for individual i
`beta`	K x 1 vector with model parameters
`alt_idx`	sum(M) x 1 vector with indices of alternatives within each choice set; 1-based indexing
`M`	N x 1 vector with number of alternatives for each individual
`weights`	N x 1 vector with weights for each observation
`include_outside_option`	whether to include outside option normalized to 0 (if so, the outside option is not included in the data)
`tol`	convergence tolerance
`max_iter`	maximum number of iterations

Value

vector with contraction's delta (ASCs) output

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
beta <- coef(fit)[fit$param_map$beta]
delta <- blp_contraction(rep(0, J), rep(1/J, J), fit$data$X,
  beta, fit$data$alt_idx, fit$data$M, fit$data$weights)
delta
```

---

coef.choicer_fit *Extract coefficients from a choicer_fit object*

Description

Extract coefficients from a choicer_fit object

Usage

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

Arguments

object	A choicer_fit object.
...	Additional arguments (ignored).

Value

Named numeric vector of estimated coefficients.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
coef(fit)
```

coef.choicer_hb *Extract posterior means from a hierarchical Bayes fit*

Description

Extract posterior means from a hierarchical Bayes fit

Usage

```
## S3 method for class 'choicer_hb'
coef(object, component = c("beta", "theta", "delta", "xi"), ...)
```

Arguments

object	A choicer_hmnl or choicer_hmnp object.
component	Which block to return: "beta" (population means b, default), "theta" (delta mean function), "delta" (alternative effects), or "xi" (unobserved quality, delta - z'theta).
...	Additional arguments (ignored).

Value

Named numeric vector of posterior means.

Examples

```
sim <- simulate_hmnl_data(N = 50, T = 2, J = 3, seed = 42)
fit <- suppressWarnings(run_hmnlogit(sim$data, "task", "alt", "choice", c("x1", "x2"),
                   person_col = "pid",
                   mcmc = list(R = 300, burn = 100)))
coef(fit)
coef(fit, component = "delta")
```

coef.choicer_mnp *Extract coefficients from a choicer_mnp object*

Description

Returns the posterior means of the identified coefficients ($\beta/\sqrt{\sigma_{11}}$, computed per draw).

Usage

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

Arguments

`object`	A choicer_mnp object.
`...`	Additional arguments (ignored).

Value

Named numeric vector of posterior means.

Examples

```
library(data.table)
set.seed(42)
N <- 100; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnprobit(dt, "id", "alt", "choice", c("x1", "x2"),
                    mcmc = list(R = 300, burn = 100))
coef(fit)
```

---

consumer_surplus.choicer_hmnl

Expected consumer surplus

Description

Computes the expected consumer surplus per choice situation (Train 2009, Ch. 3):

$$E[CS_i] = \frac{logsum_i}{-\alpha},$$

where $logsum_i$ is the expected maximum utility (see `logsum`) and α is the (fixed) price coefficient, so that $-\alpha$ is the marginal utility of income. The formula assumes *no income effects*: utility is linear in price, and the marginal utility of income is constant across the price changes considered.

Usage

```
## S3 method for class 'choicer_hmnl'
consumer_surplus(
  object,
  price_var,
  newdata = NULL,
  level = 0.95,
  weights = NULL,
  n_draws = 200L,
  ...
```

```
)

## S3 method for class 'choicer_hmnp'
consumer_surplus(
  object,
  price_var,
  newdata = NULL,
  level = 0.95,
  weights = NULL,
  ...
)

consumer_surplus(
  object,
  price_var,
  newdata = NULL,
  level = 0.95,
  weights = NULL,
  ...
)

## S3 method for class 'choicer_mnl'
consumer_surplus(
  object,
  price_var,
  newdata = NULL,
  level = 0.95,
  weights = NULL,
  ...
)

## S3 method for class 'choicer_mxl'
consumer_surplus(
  object,
  price_var,
  newdata = NULL,
  level = 0.95,
  weights = NULL,
  ...
)

## S3 method for class 'choicer_nl'
consumer_surplus(
  object,
  price_var,
  newdata = NULL,
  level = 0.95,
  weights = NULL,
```

```
  ...
)
```

Arguments

object	A fitted model object (choicer_mnl, choicer_mxl, or choicer_nl).
price_var	Name of the price variable. Must be a fixed-coefficient variable (a column of the design matrix X).
newdata	Optional counterfactual data (data.frame or list), as in logsum and predict(). When NULL (default), the data stored at fit time is used (requires keep_data = TRUE).
level	Confidence level for the normal-approximation interval around the mean CS (MNL only). Default 0.95.
weights	Optional numeric vector with one weight per choice situation, used for the mean CS (and its SE), as in predict(): for a data.frame newdata, one weight per id in order of first appearance. Defaults to equal weights. Ignored when newdata is NULL (the stored fit weights apply).
n_draws	Number of posterior draws to integrate over (hierarchical Bayes methods).
...	Additional arguments passed to methods.

Details

Consumer surplus *levels* inherit the additive utility normalization (in particular the ASC normalization), so the level is only defined up to a constant; *differences* in CS between scenarios — e.g. consumer_surplus(fit, "price", newdata = scenario) minus the baseline — are the economically meaningful quantity.

For MNL fits, a delta-method standard error of the weighted mean CS is reported (weights are the stored fit weights, or the resolved newdata weights). For MXL and NL fits only point estimates are returned (se_mean_cs = NA): the delta method for the simulated MXL logsum and the nested logsum is deferred; simulation-based intervals (Krinsky-Robb: resample coefficients from their asymptotic distribution and recompute the mean CS) are a practical alternative.

The price variable must have a *fixed* coefficient. For mixed logit a random price coefficient is rejected (as in wtp): with a random denominator $1/(-\alpha)$ generally has no finite moments.

Value

A choicer_cs object: a list with cs (per-choice- situation surplus, length N), mean_cs (weighted mean), se_mean_cs (delta-method SE; NA for MXL/NL or when the variance-covariance matrix is unavailable), ci (confidence interval for the mean), price_var, level, and n.

Methods (by class)

- consumer_surplus(choicer_hmnl): Posterior consumer surplus for the hierarchical logit: per-task logsum divided by the (positive) marginal utility of income $-\bar{\gamma}_{price}$, per posterior draw. With newdata, the return also carries the compensating variation against the estimation data (attr(, "cv")), i.e. the posterior of $(\text{logsum}_{new} - \text{logsum}_{base})/(-\bar{\gamma}_{price})$ summed over tasks. Requires a fixed-sign price coefficient; the posterior-median ratio discipline of

wtp.choicer_hb() applies. When newdata is supplied, weights is an optional non-negative length-n_tasks vector used in the aggregate CV; equal task weights are the default. The counterfactual must contain the same choice situations as the estimation data, identified by (person_col, id_col) (or id_col when person_col = NULL); rows and tasks may be reordered. Unnamed weights follow the baseline tasks' sorted (person_col, id_col) order used by the prepared data.

- consumer_surplus(choicer_hmnp): Not available for the probit (see logsum.choicer_hmnp()); roadmapped via simulated Emax.

References

Train, K. (2009). *Discrete Choice Methods with Simulation*, 2nd ed., Ch. 3. Cambridge University Press.

See Also

logsum, wtp

Examples

```
library(data.table)
sim <- simulate_mnl_data(N = 1000, J = 3, beta = c(0.8, -0.6), seed = 123,
                         outside_option = FALSE, vary_choice_set = FALSE)
fit <- run_mnlogit(sim$data, "id", "alt", "choice", c("x1", "x2"))

# treat x2 as the price variable
cs0 <- consumer_surplus(fit, price_var = "x2")
cs0

# Change in consumer surplus from a price increase on alternative 2:
# levels depend on the ASC normalization, differences do not.
dt_cf <- copy(sim$data)[alt == 2, x2 := x2 + 0.5]
cs1 <- consumer_surplus(fit, price_var = "x2", newdata = dt_cf)
delta_cs <- cs1$mean_cs - cs0$mean_cs
delta_cs  # negative: the price increase lowers expected surplus
```

---

diversion_ratios.choicer_hb

Compute aggregate diversion ratios

Description

Computes a J x J matrix of diversion ratios. Entry (i, j) is the fraction of demand lost by alternative j that is captured by alternative i when alternative j becomes less attractive.

Usage

```
## S3 method for class 'choicer_hb'
diversion_ratios(object, elast_var, eps = 0.01, n_draws = 100L, ...)

diversion_ratios(object, ...)
```

Arguments

object	A fitted model object.
elast_var	Structural covariate to perturb (hierarchical Bayes methods).
eps	Relative perturbation size (default 0.01).
n_draws	Number of posterior draws to integrate over.
...	Additional arguments passed to methods.

Value

A J x J diversion ratio matrix with alternative labels.

Methods (by class)

- `diversion_ratios(choicer_hb)`: Posterior-mean diversion ratios for hierarchical Bayes fits, from the same perturbation engine as `elasticities.choicer_hb()`: $DR(j \to k)$ is the fraction of the share alternative j loses (when its `elast_var` worsens) that flows to k — including the outside option. Columns are the perturbed alternative j, rows the receiving alternative k; the diagonal is 0.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
diversion_ratios(fit)
```

---

diversion_ratios.choicer_mnl

Diversion ratios for multinomial logit model

Description

Diversion ratios for multinomial logit model

Usage

```
## S3 method for class 'choicer_mnl'
diversion_ratios(object, ...)
```

Arguments

object	A choicer_mnl object fitted with keep_data = TRUE.
...	Additional arguments (ignored).

Value

A J x J diversion ratio matrix with alternative labels.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
diversion_ratios(fit)
```

---

diversion_ratios.choicer_mxl

Diversion ratios for mixed logit model

Description

Computes the attribute-based diversion ratio matrix. Entry (k, j) is the fraction of demand lost by alternative j that is captured by alternative k when a marginal change in alternative j's wrt_var attribute reduces s_j.

Usage

```
## S3 method for class 'choicer_mxl'
diversion_ratios(object, wrt_var, is_random_coef = FALSE, ...)
```

Arguments

object	A choicer_mxl object fitted with keep_data = TRUE.
wrt_var	Variable used to perturb alternative j's utility: a column name (character) or 1-based index. Indexes into X columns for fixed coefficients, or W columns for random coefficients (when is_random_coef = TRUE).

`is_random_coef` Logical. `TRUE` if the variable has a random coefficient (is in W), `FALSE` if fixed (in X). Default `FALSE`.

`...` Additional arguments (ignored).

Details

Unlike MNL, the MXL diversion ratio depends on which variable is perturbed: the realised coefficient β_{ik}^s varies across individuals and draws and does not cancel in the ratio. For a variable with a fixed coefficient the result is independent of the variable (β cancels); for a random-coefficient variable it is not.

Value

A J x J diversion ratio matrix with alternative labels. Cross-products are averaged across simulation draws inside the integration to avoid Jensen-style bias.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), w1 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mxlogit(
  data = dt, id_col = "id", alt_col = "alt", choice_col = "choice",
  covariate_cols = "x1", random_var_cols = "w1", S = 50L
)
diversion_ratios(fit, "x1")
diversion_ratios(fit, "w1", is_random_coef = TRUE)
```

---

diversion_ratios.choicer_nl

Diversion ratios for nested logit model

Description

Diversion ratios for nested logit model

Usage

```
## S3 method for class 'choicer_nl'
diversion_ratios(object, ...)
```

Arguments

object	A choicer_nl object fitted with keep_data = TRUE.
...	Additional arguments (ignored).

Value

A J x J diversion ratio matrix with alternative labels.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 4
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, nest := rep(c(1L, 1L, 2L, 2L), N)]
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_nestlogit(dt, "id", "alt", "choice", c("x1", "x2"), "nest")
diversion_ratios(fit)
```

---

elasticities.choicer_hb

Compute aggregate elasticities

Description

Computes a J x J matrix of aggregate elasticities. Entry (i, j) is the percentage change in the probability of choosing alternative i when the attribute of alternative j changes by 1\

Usage

```
## S3 method for class 'choicer_hb'
elasticities(object, elast_var, eps = 0.01, n_draws = 100L, ...)

elasticities(object, ...)
```

Arguments

object	A fitted model object.
elast_var	Structural covariate to perturb (hierarchical Bayes methods).
eps	Relative perturbation size (default 0.01).
n_draws	Number of posterior draws to integrate over.
...	Additional arguments passed to methods.

Value

A J x J elasticity matrix with alternative labels.

Methods (by class)

- `elasticities(choicer_hb)`: Posterior-mean aggregate arc elasticities for hierarchical Bayes fits: each inside alternative's `elast_var` is perturbed by `eps` (default 1%) and shares are recomputed per posterior draw with a common random-coefficient path, giving $E_{jk} = (\Delta s_j / s_j)/\epsilon$. Rows are responding alternatives (including the outside option), columns the perturbed alternative.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
elasticities(fit, "x1")
```

---

elasticities.choicer_mnl

Elasticities for multinomial logit model

Description

Elasticities for multinomial logit model

Usage

```
## S3 method for class 'choicer_mnl'
elasticities(object, elast_var, ...)
```

Arguments

`object`	A `choicer_mnl` object fitted with `keep_data = TRUE`.
`elast_var`	Variable for elasticity computation: a column name (character) or 1-based index into the design matrix X.
`...`	Additional arguments (ignored).

Value

A J x J elasticity matrix with alternative labels.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
elasticities(fit, "x1")
```

`elasticities.choicer_mxl` *Elasticities for mixed logit model*

Description

Elasticities for mixed logit model

Usage

```
## S3 method for class 'choicer_mxl'
elasticities(object, elast_var, is_random_coef = FALSE, ...)
```

Arguments

`object`	A `choicer_mxl` object fitted with `keep_data = TRUE`.
`elast_var`	Variable for elasticity computation: a column name (character) or 1-based index. Indexes into X columns for fixed coefficients, or W columns for random coefficients (when `is_random_coef = TRUE`).
`is_random_coef`	Logical. `TRUE` if the variable has a random coefficient (is in W), `FALSE` if fixed (in X). Default `FALSE`.
`...`	Additional arguments (ignored).

Value

A J x J elasticity matrix with alternative labels.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), w1 = rnorm(.N))]
dt[, choice := 0L]
```

```
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mxlogit(
  data = dt, id_col = "id", alt_col = "alt", choice_col = "choice",
  covariate_cols = "x1", random_var_cols = "w1", S = 50L
)
elasticities(fit, "x1")
elasticities(fit, "w1", is_random_coef = TRUE)
```

`elasticities.choicer_nl` *Elasticities for nested logit model*

Description

Elasticities for nested logit model

Usage

```
## S3 method for class 'choicer_nl'
elasticities(object, elast_var, ...)
```

Arguments

`object`	A `choicer_nl` object fitted with `keep_data = TRUE`.
`elast_var`	Variable for elasticity computation: a column name (character) or 1-based index into the design matrix X.
`...`	Additional arguments (ignored).

Value

A J x J elasticity matrix with alternative labels.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 4
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, nest := rep(c(1L, 1L, 2L, 2L), N)]
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_nestlogit(dt, "id", "alt", "choice", c("x1", "x2"), "nest")
elasticities(fit, "x1")
```

ess *Rank-normalized effective sample size (bulk and tail)*

Description

Computes the rank-normalized bulk and tail effective sample size (ESS) of Vehtari, Gelman, Simpson, Carpenter & Buerkner (2021, *Bayesian Analysis*) for each column of a matrix of posterior draws. `ess_bulk` measures the effective number of independent draws available for estimating the posterior mean/median; `ess_tail` is the minimum of the 5th- and 95th-percentile indicator ESS, relevant for tail-quantile / credible-interval precision. Autocovariance is computed via `fft` (Geyer's initial monotone sequence estimator of the integrated autocorrelation time).

Usage

```
ess(draws)
```

Arguments

`draws`	A matrix of posterior draws (rows = iterations, columns = parameters) for a single chain, or a list of such matrices (one per chain, identical dimensions).

Value

A numeric matrix, one row per parameter, two columns (`"bulk"`, `"tail"`); `NA` for parameters with zero variance or a non-finite result.

Examples

```
set.seed(42)
draws <- matrix(rnorm(2000), ncol = 2,
                dimnames = list(NULL, c("a", "b")))
ess(draws)
```

get_halton_normals *Halton draws for mixed logit*

Description

Create halton normal draws in appropriate format for mixed logit estimation

Usage

```
get_halton_normals(S, N, K_w)
```

Arguments

S	Number of draws for each choice situation
N	number of choice situations
K_w	dimension of random coefficients (number of columns in W matrix)

Value

K_w x S x N array with halton standard normal draws

Examples

```
draws <- get_halton_normals(S = 50, N = 10, K_w = 2)
dim(draws)  # 2 x 50 x 10
```

---

gof *Goodness of fit for a fitted choice model*

Description

Computes McFadden's pseudo R-squared (plain and adjusted) and the in-sample hit rate for a fitted model.

Usage

```
gof(object, null = c("equal_shares", "market_shares"), ...)

## S3 method for class 'choicer_fit'
gof(object, null = c("equal_shares", "market_shares"), ...)
```

Arguments

object	A fitted model object (choicer_mnl, choicer_mxl, or choicer_nl).
null	Null model for the pseudo R-squared: "equal_shares" (default) or "market_shares".
...	Additional arguments passed to methods.

Details

Two null models are available for the pseudo R-squared $R^2 = 1 - LL/LL_0$ (adjusted: $R^2_{adj} = 1 - (LL - K)/LL_0$ with K the number of estimated parameters):

- "equal_shares" (default): every alternative in individual i's choice set is equally likely, so $LL_0 = -\sum_i w_i \log(M_i + 1_{outside})$. This is exact for unbalanced choice sets and arbitrary weights.
- "market_shares": the maximized log-likelihood of an ASC-only model, $LL_0 = \sum_j N_j \log(s_j)$ with N_j the choice counts and s_j the observed market shares (including the outside option when present). This closed form is valid only for balanced choice sets and uniform weights; otherwise an error suggests refitting an ASC-only model.

The hit rate is the weighted share of individuals whose observed choice has the highest predicted probability. When the model includes an outside option, the outside good competes for the predicted maximum (its probability is $1 - \sum_j p_{ij}$), and an individual predicted to choose the outside good is a hit when they actually did.

Both the null log-likelihood and the hit rate require the stored estimation data; models fitted with `keep_data = FALSE` return NA fields with a message.

Value

A `choicer_gof` object: a list with `loglik`, `loglik_null`, `null`, `mcfadden_r2`, `mcfadden_r2_adj`, `hit_rate`, `nobs`, and `n_params`.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
gof(fit)
gof(fit, null = "market_shares")
```

---

logLik.choicer_fit *Extract log-likelihood from a choicer_fit object*

Description

Returns a logLik object, which enables AIC() and BIC() automatically.

Usage

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

Arguments

`object`	A choicer_fit object.
`...`	Additional arguments (ignored).

Value

A logLik object with df and nobs attributes.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
logLik(fit)
AIC(fit)
BIC(fit)
```

---

logsum.choicer_hmnl *Expected logsum (inclusive value) per choice situation*

Description

Computes the expected maximum utility ("logsum" or inclusive value) for each choice situation, up to the additive constant of the extreme-value error:

- MNL: $\log \sum_j \exp(V_{ij})$.
- MXL: $E_\beta[\log \sum_j \exp(V_{ij}(\beta))]$, simulated by averaging the log-sum-exp *across* the deterministic Halton draws (regenerated from `object$draws_info`). Taking the log-sum-exp of draw-averaged utilities would understate the expectation (Jensen's inequality), so a dedicated kernel (`mxl_logsum`) is used.
- NL: $\log \sum_b \exp(\lambda_b I_{ib})$ with the nest inclusive value $I_{ib} = \log \sum_{j \in b} \exp(V_{ij}/\lambda_b)$ (singleton nests have $\lambda_b = 1$).

When the model includes an outside option, its normalized utility $V = 0$ contributes an $\exp(0)$ term to the sum.

Usage

```
## S3 method for class 'choicer_hmnl'
logsum(object, newdata = NULL, n_draws = 200L, ...)

## S3 method for class 'choicer_hmnp'
logsum(object, newdata = NULL, ...)

logsum(object, newdata = NULL, ...)

## S3 method for class 'choicer_mnl'
logsum(object, newdata = NULL, ...)

## S3 method for class 'choicer_mxl'
```

```
logsum(object, newdata = NULL, ...)

## S3 method for class 'choicer_nl'
logsum(object, newdata = NULL, ...)
```

Arguments

object	A fitted model object (choicer_mnl, choicer_mxl, or choicer_nl).
newdata	Optional counterfactual data: a data.frame in the fit-time long format or a list with X, alt_idx, M (plus W for MXL), as in predict(). When NULL (default), the data stored at fit time is used (requires keep_data = TRUE).
n_draws	Number of posterior draws to integrate over (hierarchical Bayes methods; thinned evenly from the kept draws).
...	Additional arguments passed to methods.

Details

Logsum *levels* depend on the ASC normalization (and, more generally, on any additive utility normalization), so only logsum *differences* between scenarios (e.g. via newdata) are meaningful.

Value

Numeric vector with one logsum per choice situation. With a data.frame newdata, choice situations are ordered by id (as in predict()).

Methods (by class)

- logsum(choicer_hmnl): Posterior expected logsum for the hierarchical logit: per choice situation, $\log(1+\sum_j \exp V_j)$ against the outside-option anchor, averaged over posterior draws with one $\beta \sim N(b_r, W_r)$ draw each. Returns the per-task posterior mean vector.
- logsum(choicer_hmnp): The probit expected maximum has no closed form; simulated-Emax surplus for the HMNP is on the roadmap.

See Also

consumer_surplus

Examples

```
library(data.table)
sim <- simulate_mnl_data(N = 500, J = 3, beta = c(0.8, -0.6), seed = 1,
                         outside_option = FALSE, vary_choice_set = FALSE)
fit <- run_mnlogit(sim$data, "id", "alt", "choice", c("x1", "x2"))
head(logsum(fit))
```

`mc_asymptotics` *Asymptotic diagnostics for a Monte Carlo study*

Description

Consumes a `choicer_mc` object and returns per-parameter asymptotic diagnostics: Monte Carlo bias (with MC standard error), empirical SD of the estimates, mean of the reported standard errors, SE-to-SD ratio (information-matrix-equality check), Wald coverage at nominal 90 / 95 / 99 percent with Wilson confidence bands, moments of the studentized statistic `z = (theta_hat - theta_0) / se`, and four normality tests on `z` (Shapiro-Wilk, Anderson-Darling via `goftest::ad.test`, a hand-coded Jarque-Bera statistic, and a one-sample Kolmogorov-Smirnov test against `N(0, 1)`).

Usage

```
mc_asymptotics(
  mc,
  level = 0.95,
  se_col = "se",
  conv_threshold = 0.99,
  se_ratio_threshold_floor = 0.1
)
```

Arguments

`mc`	A `choicer_mc` object returned by `monte_carlo()`.
`level`	Confidence level for the Wilson bands on coverage rates. Defaults to `0.95`.
`se_col`	Name of the column in `mc$replications` to use as the standard-error source. Defaults to `"se"` (the Hessian-based SE stored by `monte_carlo()`). Callers that augment replications with an alternative SE flavor (e.g., `"se_bhhh"` for a BHHH/OPG comparison) can pass that column name to recompute every SE-dependent diagnostic (`mean_se`, `se_ratio`, `mean_se_w`, `cov90/95/99`, z-moments, normality tests, pass flags) against that flavor. Useful for the information-matrix-equality check in Claim 4 of the MXL validation suite.
`conv_threshold`	Numeric in `[0, 1]`. Minimum fraction of replications that must converge for the per-parameter `pass_convergence` flag to be `TRUE`. The flag compares `R_used / R_total` (per parameter) against this threshold. Defaults to `0.99`.
`se_ratio_threshold_floor`	Numeric scalar. Minimum half-width for the `pass_se_ratio` band. The actual band used is `max(se_ratio_threshold_floor, 3 * 1.4 / sqrt(R_used))`, where the `1.4 / sqrt(R)` term approximates the large-sample SD of `mean_se / sd_emp`. The floor guarantees the band is never tighter than the historical hard cutoff. Defaults to `0.10`.

Details

Six logical pass / fail flags are attached to every parameter row: `pass_bias` requires `|bias_mc_se| < 3`; `pass_se_ratio` requires `|se_ratio - 1|` to lie within `max(se_ratio_threshold_floor, 3 * 1.4 / sqrt(R_used))` (a noise-aware band that widens at small `R_used` and tightens to the floor at large `R_used`); `pass_cov95` requires the nominal 95 percent level to lie in the Wilson band for empirical coverage; `pass_skew` requires `|skew_z| < 0.3`; `pass_kurt` requires excess kurtosis of `z` in `[-0.5, 1.0]`; `pass_convergence` requires the per-parameter convergence rate (`R_used / R_total`) to meet `conv_threshold`.

Non-converged replications are excluded per parameter (reported in `R_excluded`). Winsorized (5 percent / 95 percent) versions of `bias`, `sd_emp`, and `mean_se` are reported in parallel columns (`bias_w`, `sd_emp_w`, `mean_se_w`) so silent outlier exclusion is transparent to the reader. Two robust SE-to-SD ratios accompany the Hessian-mean-based `se_ratio`: `se_ratio_med` (median SE divided by the empirical SD) and `se_ratio_w` (winsorized mean SE divided by the winsorized empirical SD); both stay near 1 when 1-2 replications produce near-singular Hessians that inflate `mean_se`. The companion `se_med` column reports the median per-replication SE used by `se_ratio_med`. Neither robust ratio drives a `pass_*` flag — they are purely informational.

Winsorized z-moment counterparts (`mean_z_w`, `sd_z_w`, `skew_z_w`, `kurt_excess_z_w`) are reported alongside the raw z-moments and feed an additional `pass_z_w` flag (Winsorized skew within the same band as `pass_skew` AND Winsorized excess kurtosis within the same band as `pass_kurt`). A companion `pass_cov95_w` flag is `TRUE` when either `pass_cov95` is `TRUE` OR the per-rep Winsorized z-CI (the empirical 2.5 / 97.5 percentiles of the Winsorized z) covers truth-zero. These two flags are designed for boundary scenarios (e.g., near-zero variance components) where a small number of reps with vanishing SE inflate the raw z-moments without indicating an estimator defect.

Value

An object of class `choicer_mc_asymptotics` — a `data.table` with one row per unique parameter and columns documented above — with `meta` attached as an attribute (`attr(x, "meta")`).

Examples

```
sim_fun <- function(seed) simulate_mnl_data(N = 1000, J = 3, seed = seed)
fit_fun <- function(sim) run_mnlogit(
  data = sim$data, id_col = "id", alt_col = "alt", choice_col = "choice",
  covariate_cols = c("x1", "x2"), outside_opt_label = 0L,
  include_outside_option = FALSE, use_asc = TRUE,
  control = list(print_level = 0L)
)
mc <- monte_carlo(sim_fun, fit_fun, R = 50L, seed = 1L, progress = FALSE)
mc_asymptotics(mc)
```

`mcse` *Monte Carlo standard error of posterior summaries*

Description

Monte Carlo standard error (MCSE) of the posterior mean or median, using the rank-normalized bulk/tail ESS from ess. `kind = "mean"`: `MCSE = SD / sqrt(ess_bulk)`. `kind = "median"`: `MCSE = sqrt(pi/2) * SD / sqrt(ess_tail)` (the closed-form normal asymptotic-efficiency approximation; `2/pi` is the asymptotic relative efficiency of the sample median vs. the mean under normality).

Usage

```
mcse(draws, kind = c("mean", "median"))
```

Arguments

`draws`	A matrix of posterior draws (rows = iterations, columns = parameters) for a single chain, or a list of such matrices (one per chain, identical dimensions).
`kind`	`"mean"` (default) or `"median"`.

Value

Named numeric vector, one value per parameter (`NA` when the underlying ESS or pooled SD is undefined).

Examples

```
set.seed(42)
draws <- matrix(rnorm(2000), ncol = 2,
                dimnames = list(NULL, c("a", "b")))
mcse(draws)
mcse(draws, kind = "median")
```

`mode_choice` *Intercity travel mode choice*

Description

Stated choices of intercity travel mode for 210 travellers, each choosing among the same four modes: air, train, bus and car. This is the classic Greene & Hensher (1997) data set, reshaped into choicer's long layout (one row per traveller-by-alternative). It is a convenient, recognizable example for multinomial and nested logit models and the demand/welfare toolkit (elasticities, diversion ratios, willingness-to-pay, counterfactuals).

Usage

```
mode_choice
```

Format

A data frame with 840 rows (210 travellers x 4 modes) and 9 columns:

id Integer traveller (choice situation) identifier, 1-210.

mode Factor giving the travel mode: `"air"`, `"train"`, `"bus"` or `"car"`. Use as the alternative column.

choice Integer indicator, 1 for the chosen mode and 0 otherwise. Exactly one mode is chosen per traveller.

wait Terminal waiting time in minutes (0 for car).

travel In-vehicle travel time in minutes.

vcost In-vehicle cost component, in currency units.

gcost Generalized cost measure, in currency units.

income Household income (traveller level, in thousands).

size Size of the travelling party (traveller level).

Details

`wait`, `travel`, `vcost` and `gcost` vary across modes within a traveller, while `income` and `size` are traveller-level attributes that are constant across modes. A standard specification regresses the choice on `wait`, `travel` and `vcost` with alternative-specific constants; `vcost` then plays the role of price for willingness-to-pay and consumer-surplus calculations.

The sample is *choice-based*: the survey over-sampled the less popular modes (air, train, bus) and under-sampled car, so sample choice shares do not estimate population mode shares. With a full set of alternative-specific constants the slope coefficients remain consistently estimated under this design (Manski and Lerman, 1977), and willingness-to-pay ratios are unaffected; the constants, and any shares, elasticities or surplus levels computed from fitted probabilities, inherit the sampling design. To target population quantities, attach WESML weights with `wesml_weights` using external population shares; see `vignette("wesml", package = "choicer")`.

Source

Greene, W. H. and Hensher, D. A. (1997). Reshaped from the `TravelMode` data distributed with the **AER** package (`https://CRAN.R-project.org/package=AER`). The same data appear in Greene's *Econometric Analysis* and in several other choice-modelling packages.

References

Manski, C. F. and Lerman, S. R. (1977). The estimation of choice probabilities from choice based samples. *Econometrica*, 45(8), 1977-1988.

Examples

```
data(mode_choice)
head(mode_choice)
table(mode_choice$mode[mode_choice$choice == 1L])
```

monte_carlo *Monte Carlo parameter recovery*

Description

Replicates a (DGP -> fit) cycle R times with independent seeds and collects per-parameter estimates, standard errors, bias, and coverage. Returns a choicer_mc object; call summary() for aggregated statistics (mean estimate, bias, RMSE, coverage rate, convergence rate).

Usage

```
monte_carlo(
  sim_fun,
  fit_fun,
  R = 100,
  seed = 1L,
  parallel = FALSE,
  progress = TRUE,
  ...
)
```

Arguments

sim_fun	Function of seed returning a choicer_sim.
fit_fun	Function of a choicer_sim returning a choicer_fit.
R	Number of replications.
seed	Base integer seed. Replication r uses seed + r - 1L.
parallel	Logical; if TRUE and future.apply is available, run replications in parallel using the user's active future::plan().
progress	Logical; print a one-line progress update per iteration in serial mode. Ignored when parallel = TRUE.
...	Unused.

Details

Each iteration calls sim_fun(seed = seed + r - 1L), then fit_fun(sim). Write sim_fun as a closure that captures N, J, and other DGP settings and forwards seed. Write fit_fun as a closure that takes a choicer_sim and returns a fitted choicer_fit object, wrapping any data-preparation, draws, or optimizer-control setup.

Value

A choicer_mc object: a list with elements replications (a long data.table with one row per estimated parameter per replication) and meta (run metadata).

Examples

```
sim_fun <- function(seed) simulate_mnl_data(N = 1000, J = 4, seed = seed)
fit_fun <- function(sim) run_mnlogit(
  data = sim$data, id_col = "id", alt_col = "alt", choice_col = "choice",
  covariate_cols = c("x1", "x2"), outside_opt_label = 0L,
  include_outside_option = FALSE, use_asc = TRUE,
  control = list(print_level = 0L)
)
mc <- monte_carlo(sim_fun, fit_fun, R = 5, seed = 1L, progress = FALSE)
summary(mc)
```

---

mxl_blp_contraction *BLP contraction mapping for mixed logit*

Description

Finds the ASC (delta) parameters such that predicted market shares match target shares, using the contraction mapping of Berry, Levinsohn, and Pakes (1995).

Usage

```
mxl_blp_contraction(
  delta,
  target_shares,
  X,
  W,
  beta,
  mu,
  L_params,
  alt_idx,
  M,
  weights,
  eta_draws,
  rc_dist,
  rc_correlation = TRUE,
  rc_mean = FALSE,
  include_outside_option = FALSE,
  tol = 1e-08,
  max_iter = 1000L,
  gen_seed = -1L,
  gen_scramble = 1L,
  gen_S = 0L
)
```

Arguments

delta	J-1 or J vector with initial guess for deltas (ASCs)
target_shares	J vector with target market shares
X	design matrix for fixed coefficients; sum(M_i) x K_x
W	design matrix for random coefficients; sum(M_i) x K_w or J x K_w
beta	K_x vector with fixed coefficients
mu	K_w vector with mean parameters (raw, will be transformed if log-normal)
L_params	Cholesky parameters vector
alt_idx	sum(M) x 1 vector with indices of alternatives; 1-based indexing
M	N x 1 vector with number of alternatives for each individual
weights	N x 1 vector with weights for each observation
eta_draws	Array with draws; K_w x S x N
rc_dist	K_w vector indicating distribution (0=normal, 1=log-normal)
rc_correlation	whether random coefficients are correlated
rc_mean	whether mu parameters represent means (TRUE) or are zero (FALSE)
include_outside_option	whether outside option is included
tol	convergence tolerance (default 1e-8)
max_iter	maximum iterations (default 1000)
gen_seed	Integer master seed for the on-the-fly Halton generator. `< 0` (default) uses the materialized `eta_draws` cube; `>= 0` generates draws on the fly from this seed.
gen_scramble	Integer scramble mode for on-the-fly generation: `0` = identity permutations, `1` = seeded position-wise digit permutations.
gen_S	Integer number of draws per individual, used only when `gen_seed >= 0`.

Value

vector with converged delta (ASC) values

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), w1 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
d <- prepare_mxl_data(dt, "id", "alt", "choice", "x1", "w1")
eta <- get_halton_normals(50, d$N, ncol(d$W))
fit <- run_mxlogit(input_data = d, eta_draws = eta)
pm <- fit$param_map
delta <- mxl_blp_contraction(rep(0, J), rep(1/J, J), d$X, d$W,
  coef(fit)[pm$beta], rep(0, ncol(d$W)), coef(fit)[pm$sigma],
```

```
  d$alt_idx, d$M, d$weights, eta, rc_dist = rep(0L, ncol(d$W)),
  rc_correlation = FALSE, rc_mean = FALSE)
delta
```

`new_choicer_sim` *Construct a* `choicer_sim` *object*

Description

Wraps simulated data, true parameter values, and DGP settings into a classed list. Returned by `simulate_mnl_data()`, `simulate_mxl_data()`, and `simulate_nl_data()`, and consumed by `recovery_table()`.

Usage

```
new_choicer_sim(data, true_params, settings, model)
```

Arguments

`data`	A `data.table` of simulated choice observations.
`true_params`	Named list of true DGP parameters (e.g. `beta`, `delta`, `Sigma`, `mu`, `lambdas`).
`settings`	Named list of DGP settings (e.g. `N`, `J`, `K_x`).
`model`	Character scalar: `"mnl"`, `"mxl"`, `"nl"`, `"mnp"`, `"hmnl"`, or `"hmnp"`.

Value

A list of class `choicer_sim`.

`nl_blp_contraction` *BLP95 contraction mapping for the Nested Logit model*

Description

Damped iterative fixed point recovering delta given target shares, using the NL probability structure. `damping = 1` reproduces the plain BLP update.

Usage

```
nl_blp_contraction(
  delta,
  target_shares,
  X,
  beta,
  lambda,
  alt_idx,
  nest_idx,
  M,
  weights,
  include_outside_option = FALSE,
  damping = 1,
  tol = 1e-08,
  max_iter = 1000L
)
```

Arguments

delta	J x 1 vector with initial guess for deltas (ASCs).
target_shares	vector with target shares (outside-option share first when present).
X	sum(M) x K design matrix with covariates.
beta	K x 1 vector with fixed coefficients.
lambda	full nest dissimilarity vector of length n_nests (singletons = 1).
alt_idx	sum(M) x 1 vector with indices of alternatives; 1-based indexing.
nest_idx	J x 1 vector with nest indices for each alternative; 1-based indexing.
M	N x 1 vector with number of alternatives for each individual.
weights	N x 1 vector with weights for each observation.
include_outside_option	whether to include outside option normalized to V=0, lambda=1.
damping	damping factor for the update (default 1.0 = plain BLP).
tol	convergence tolerance.
max_iter	maximum number of iterations.

Value

vector with contraction's delta (ASCs) output.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 4
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
```

```
dt[, nest := ifelse(alt <= 2, "A", "B")]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_nestlogit(dt, "id", "alt", "choice", c("x1", "x2"), "nest")
beta <- coef(fit)[fit$param_map$beta]
lambda <- rep(1, length(unique(fit$data$nest_idx)))
lambda[as.integer(names(which(table(fit$data$nest_idx) > 1)))] <-
  coef(fit)[fit$param_map$lambda]
delta <- nl_blp_contraction(rep(0, J), rep(1/J, J), fit$data$X, beta, lambda,
  fit$data$alt_idx, fit$data$nest_idx, fit$data$M, fit$data$weights)
delta
```

`nobs.choicer_fit` *Extract number of observations from a choicer_fit object*

Description

Extract number of observations from a choicer_fit object

Usage

```
## S3 method for class 'choicer_fit'
nobs(object, ...)
```

Arguments

`object`	A choicer_fit object.
`...`	Additional arguments (ignored).

Value

Integer number of choice situations.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
nobs(fit)
```

nobs.choicer_hb *Number of choice situations behind a hierarchical Bayes fit*

Description

Number of choice situations behind a hierarchical Bayes fit

Usage

```
## S3 method for class 'choicer_hb'
nobs(object, ...)
```

Arguments

object	A choicer_hmnl or choicer_hmnp object.
...	Additional arguments (ignored).

Value

Integer count of choice situations (tasks).

nobs.choicer_mnp *Extract number of observations from a choicer_mnp object*

Description

Extract number of observations from a choicer_mnp object

Usage

```
## S3 method for class 'choicer_mnp'
nobs(object, ...)
```

Arguments

object	A choicer_mnp object.
...	Additional arguments (ignored).

Value

Integer number of choice situations.

Examples

```
library(data.table)
set.seed(42)
N <- 100; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnprobit(dt, "id", "alt", "choice", c("x1", "x2"),
                    mcmc = list(R = 300, burn = 100))
nobs(fit)
```

---

ppc_shares *Posterior-predictive share check for hierarchical Bayes fits*

Description

Compares each alternative's observed take rate (share of choice situations in which it was chosen, including the outside option) with its posterior-predictive share from `predict.choicer_hb()`. Large systematic gaps indicate model misfit — e.g. a missing covariate or an outside-option share the delta level cannot rationalize.

Usage

```
ppc_shares(object, n_draws = 200L)
```

Arguments

`object`	A `choicer_hmnl` or `choicer_hmnp` fit (with `keep_data = TRUE`).
`n_draws`	Posterior draws to integrate over (default 200).

Value

A `data.table` with columns `alternative`, `observed`, `predicted`, `lower`, `upper` (95% posterior-predictive interval), and `covered` (is the observed share inside the interval).

Examples

```
sim <- simulate_hmnl_data(N = 100, T = 3, J = 4, seed = 42)
fit <- suppressWarnings(run_hmnlogit(sim$data, "task", "alt", "choice", c("x1", "x2"),
                    person_col = "pid",
                    mcmc = list(R = 500, burn = 200)))
ppc_shares(fit)
```

`predict.choicer_hb` *Posterior choice probabilities and shares for hierarchical Bayes fits*

Description

Computes counterfactual choice probabilities, integrating over the posterior draws and (at the population level) over the random-coefficient distribution: for each kept draw $(b_r, W_r, \delta_r, \ldots)$ one $\beta \sim N(b_r, W_r)$ is drawn and the model probabilities are averaged. Alternatives in `newdata` that were not in the estimation sample receive a posterior-predictive $\delta_{new} \sim N(z'_{new}\theta_r, \sigma^2_{d,r})$ — the entry counterfactual unlocked by the random-effects δ. Price or subsidy counterfactuals are just modified covariate columns in `newdata`.

Usage

```
## S3 method for class 'choicer_hb'
predict(
  object,
  newdata = NULL,
  level = c("population", "individual"),
  n_draws = 200L,
  aggregate = TRUE,
  ...
)
```

Arguments

`object`	A `choicer_hmnl` or `choicer_hmnp` fit.
`newdata`	Data frame with the estimation columns (choice column not required). `NULL` (default) predicts on the estimation data.
`level`	`"population"` (default) integrates over $N(b, W)$; `"individual"` uses the respondent-level β_i (requires the prediction rows to belong to estimation respondents; fully posterior-integrated when the fit kept `keep_beta_i = "draws"`).
`n_draws`	Number of posterior draws to integrate over (thinned evenly from the kept draws; default 200).
`aggregate`	If `TRUE` (default) return a per-alternative posterior share table (including the outside option); if `FALSE` return the posterior-mean probability per row of the prediction data.
`...`	Ignored.

Details

HMNL probabilities are closed-form logit; HMNP probabilities use the 1-D Gauss-Hermite representation of the iid-probit integral $P(j) = \int \phi(u) \prod_{k \neq j} \Phi(V_j - V_k + u) du$.

Value

With `aggregate = TRUE`, a `data.table` with columns `alternative`, `share` (posterior mean), `sd`, `lower`, `upper` (95% equal-tailed interval); the posterior share draws are attached as `attr(, "draws")`. With `aggregate = FALSE`, a numeric vector of posterior-mean choice probabilities, one per prediction row.

Examples

```
sim <- simulate_hmnl_data(N = 100, T = 3, J = 4, seed = 42)
fit <- suppressWarnings(run_hmnlogit(sim$data, "task", "alt", "choice", c("x1", "x2"),
                    person_col = "pid", alt_covariate_cols = "z1",
                    mcmc = list(R = 500, burn = 200)))
predict(fit)                        # posterior shares, estimation data
cf <- sim$data
cf$x1 <- cf$x1 + 0.5                # a counterfactual attribute change
predict(fit, newdata = cf)
```

---

predict.choicer_mnl *Predict from a multinomial logit model*

Description

Computes choice probabilities or aggregate market shares, either for the data used at fit time (default) or for counterfactual `newdata`.

Usage

```
## S3 method for class 'choicer_mnl'
predict(
  object,
  type = c("probabilities", "shares"),
  newdata = NULL,
  weights = NULL,
  ...
)
```

Arguments

`object`	A choicer_mnl object.
`type`	One of "probabilities" (individual-level choice probabilities) or "shares" (aggregate market shares).
`newdata`	Optional data for counterfactual prediction. Either: • a data.frame in the same long format used at fit time (one row per id-alternative pair, with the fit-time id, alternative, and covariate columns; a choice column is not required). Alternative labels must have been seen at fit time; per-id subsets of alternatives are allowed.

- a list with elements X, alt_idx, M (and optionally weights) matching the layout of object$data — the "modified design matrix" path for policy simulation (e.g., perturb a column of object$data$X). alt_idx must use the fit-time integer codes from object$alt_mapping.

When NULL (default), the data stored at fit time is used (requires keep_data = TRUE).

weights — Optional numeric vector with one weight per choice situation, used for type = "shares" aggregation. For a data.frame newdata, supply one weight per id in order of first appearance in newdata (weights are realigned internally to the sorted row order). Defaults to equal weights. Ignored when newdata is NULL (the stored fit weights apply).

... — Additional arguments (ignored).

Value

For "probabilities": a list with choice_prob and utility vectors. For "shares": a named numeric vector of market shares per alternative. With a data.frame newdata, rows are ordered by id, then by fit-time alternative code (alt_int in object$alt_mapping).

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
predict(fit, type = "shares")
predict(fit, type = "probabilities")

# Counterfactual: increase x1 for alternative 2
dt_cf <- copy(dt)[alt == 2, x1 := x1 + 1]
predict(fit, type = "shares", newdata = dt_cf)
```

---

predict.choicer_mxl *Predict from a mixed logit model*

Description

Computes simulated choice probabilities or aggregate market shares using deterministic Halton draws, either for the data used at fit time (default) or for counterfactual newdata.

Usage

```
## S3 method for class 'choicer_mxl'
predict(
  object,
  type = c("probabilities", "shares"),
  newdata = NULL,
  weights = NULL,
  ...
)
```

Arguments

object	A choicer_mxl object.
type	Either "probabilities" (per-observation simulated choice probabilities) or "shares" (aggregate simulated market shares).
newdata	Optional data for counterfactual prediction. Either: • a data.frame in the same long format used at fit time (one row per id-alternative pair, with the fit-time id, alternative, fixed-coefficient, and random-coefficient columns; a choice column is not required). Alternative labels must have been seen at fit time; per-id subsets of alternatives are allowed. • a list with elements X, W, alt_idx, M (and optionally weights) matching the layout of object$data — the "modified design matrix" path for policy simulation. alt_idx must use the fit-time integer codes from object$alt_mapping. When NULL (default), the data stored at fit time is used (requires keep_data = TRUE). Halton draws are regenerated deterministically from object$draws_info with one block of draws per choice situation in newdata.
weights	Optional numeric vector with one weight per choice situation, used for type = "shares" aggregation. For a data.frame newdata, supply one weight per id in order of first appearance in newdata (weights are realigned internally to the sorted row order). Defaults to equal weights. Ignored when newdata is NULL (the stored fit weights apply).
...	Additional arguments (ignored).

Value

For "probabilities": a list with choice_prob and utility vectors averaged across simulation draws. For "shares": a named numeric vector of simulated market shares per alternative. With a data.frame newdata, rows are ordered by id, then by fit-time alternative code (alt_int in object$alt_mapping).

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), w1 = rnorm(.N))]
dt[, choice := 0L]
```

```
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mxlogit(
  data = dt, id_col = "id", alt_col = "alt", choice_col = "choice",
  covariate_cols = "x1", random_var_cols = "w1", S = 50L
)
predict(fit, type = "shares")
predict(fit, type = "probabilities")
```

`predict.choicer_nl` *Predict from a nested logit model*

Description

Computes choice probabilities or aggregate market shares, either for the data used at fit time (default) or for counterfactual `newdata`.

Usage

```
## S3 method for class 'choicer_nl'
predict(
  object,
  type = c("probabilities", "shares"),
  newdata = NULL,
  weights = NULL,
  ...
)
```

Arguments

`object`	A choicer_nl object.
`type`	One of "probabilities" (individual-level choice probabilities) or "shares" (aggregate market shares).
`newdata`	Optional data for counterfactual prediction. Either: • a data.frame in the same long format used at fit time (one row per id-alternative pair, with the fit-time id, alternative, and covariate columns; a choice column is not required). Alternative labels must have been seen at fit time; per-id subsets of alternatives are allowed. The alternative-to-nest mapping always comes from the fitted object (it indexes the estimated `lambda` parameters), so a nest column in `newdata` is not required and is ignored if present. • a list with elements `X`, `alt_idx`, `M` (and optionally `weights`) matching the layout of `object$data` — the "modified design matrix" path for policy simulation. `alt_idx` must use the fit-time integer codes from `object$alt_mapping`. When `NULL` (default), the data stored at fit time is used (requires `keep_data = TRUE`).

weights	Optional numeric vector with one weight per choice situation, used for type = "shares" aggregation. For a data.frame newdata, supply one weight per id in order of first appearance in newdata (weights are realigned internally to the sorted row order). Defaults to equal weights. Ignored when newdata is NULL (the stored fit weights apply).
...	Additional arguments (ignored).

Value

For "probabilities": a list with choice_prob and utility vectors. For "shares": a named numeric vector of market shares per alternative. With a data.frame newdata, rows are ordered by id, then by fit-time alternative code (alt_int in object$alt_mapping).

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 4
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, nest := rep(c(1L, 1L, 2L, 2L), N)]
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_nestlogit(dt, "id", "alt", "choice", c("x1", "x2"), "nest")
predict(fit, type = "shares")
predict(fit, type = "probabilities")
```

---

prepare_hmnl_data *Prepare inputs for hierarchical multinomial logit estimation*

Description

Prepares and validates panel (or cross-sectional) choice data for the hierarchical Bayesian multinomial logit. The model has two random-effect levels: respondent-level structural tastes $\beta_i \sim N(b, W)$ over the covariate_cols, and a global alternative-level effect $\delta_j = z_j'\theta + \xi_j$, $\xi_j \sim N(0, \sigma_d^2)$, with mean-function design z_j built from alt_covariate_cols.

Usage

```
prepare_hmnl_data(
  data,
  id_col,
  alt_col,
  choice_col,
  covariate_cols,
  person_col = NULL,
  alt_covariate_cols = NULL,
```

```
  outside_opt_label = NULL,
  cf_residual_col = NULL,
  include_outside_option = TRUE,
  rc_dist = NULL
)
```

Arguments

`data`	Data frame containing choice data.
`id_col`	Name of the column identifying choice situations (tasks). Task ids only need to be unique within a respondent.
`alt_col`	Name of the column identifying alternatives.
`choice_col`	Name of the column indicating the chosen alternative (1 = chosen, 0 = not chosen).
`covariate_cols`	Vector of names of structural covariate columns (the random-coefficient dimensions).
`person_col`	Name of the respondent column grouping choice situations. `NULL` (default) makes each choice situation its own respondent.
`alt_covariate_cols`	Names of alternative-level covariate columns (constant within each alternative) forming the δ mean function. `NULL` (default) gives an intercept-only design (P = 1).
`outside_opt_label`	Label of physical outside-option rows, removed when `include_outside_option = TRUE` (the outside good is implicit).
`cf_residual_col`	Name of a first-stage residual column (control function for an endogenous covariate), appended to `X`. Default `NULL`.
`include_outside_option`	Logical; if `TRUE` (default) an implicit outside option with systematic utility 0 is part of every choice set.
`rc_dist`	Integer vector, one entry per column of `covariate_cols`: `0` for a normal random coefficient, `1` for log-normal (the coefficient enters utility as `exp(beta_ik)`; hierarchy normal on the log scale). Default `NULL` is all-normal. Automatically aligned through dropped columns; a `cf_residual_col` coordinate is always normal.

Details

Structure. The design matrix `X` carries structural covariates only — no alternative-specific-constant dummies. The alternative effect δ_j is indexed by `alt_of_row` (integer codes `1..J`), so memory and compute scale with the number of rows, not with `J` extra design columns.

Outside option. With `include_outside_option = TRUE` (the default) the outside good is modelled implicitly, following the `prepare_mnl_data()` convention: physical outside rows (identified by `outside_opt_label`) are removed, the estimation kernels add the outside term (systematic utility 0), and a choice situation whose inside rows are all `0` in `choice_col` is coded as "outside chosen"

(choice_pos = 0). The outside option anchors the location of δ (mean utility relative to the outside good).

Cross-section vs panel. person_col groups choice situations into respondents sharing one β_i. With person_col = NULL (default) every choice situation is its own respondent (Ti all 1) — the cross-sectional random-coefficients mode.

Control function. cf_residual_col (a user-supplied first-stage residual, Petrin & Train 2010) is appended to X as an ordinary covariate; its provenance is recorded in data_spec. The first stage is NOT run here — supplying a valid residual is the user's responsibility.

Value

A list of class c("choicer_data_hmnl", "list") containing:

- X: Structural design matrix (total_rows x K_struct), no ASC columns; cf_residual_col last when supplied.
- alt_of_row: Integer alternative code per row (1..J).
- alt_idx: Alias of alt_of_row for the pooled-MLE init.
- Z: Alternative-level design (J x P), intercept first.
- M: Inside alternatives per choice situation.
- choice_pos: 1-based within-task position of the chosen row; 0 = outside option chosen.
- Ti: Choice situations per respondent.
- person_ids, N_persons, n_tasks, J, K_struct, P.
- include_outside_option: Logical flag.
- alt_mapping: Data.table mapping alternatives to summary statistics (outside option is alt_int = 0).
- param_map: Named list of index vectors (beta, theta), robust to collinearity drops.
- rc_dist: Integer vector aligned with the columns of X.
- dropped_cols, dropped_z_cols: Dropped column names, if any.
- data_spec: Column-name metadata (incl. person_col, outside_opt_label, cf_residual_col, alt_covariate_cols).

See Also

prepare_hmnp_data() for the hierarchical probit counterpart.

Examples

```
library(data.table)
set.seed(42)
N <- 20; T <- 3; J <- 4
dt <- data.table(
  pid  = rep(1:N, each = T * J),
  task = rep(seq_len(N * T), each = J),
  alt  = rep(1:J, N * T)
)
dt[, `:=`(x1 = rnorm(.N), x2 = runif(.N, -1, 1))]
```

```
dt[, quality := 0.1 * alt]               # alternative-level covariate
dt[, choice := 0L]
# leave some tasks all-zero: outside option chosen
dt[, choice := if (runif(1) < 0.8) sample(c(1L, rep(0L, J - 1))) else 0L,
   by = task]
input <- prepare_hmnl_data(dt, "task", "alt", "choice", c("x1", "x2"),
                           person_col = "pid",
                           alt_covariate_cols = "quality")
str(input$Z)
input$alt_mapping
```

`prepare_hmnp_data` *Prepare inputs for hierarchical multinomial probit estimation*

Description

Prepares and validates panel (or cross-sectional) choice data for the hierarchical Bayesian multinomial probit with iid $N(0, \sigma^2)$ utility shocks. The model shares its two-level random-effect structure with `prepare_hmnl_data()`: respondent-level structural tastes $\beta_i \sim N(b, W)$ over the `covariate_cols` (normal only — the probit keeps full conjugacy), and a global alternative-level effect $\delta_j = z_j'\theta + \xi_j$, $\xi_j \sim N(0, \sigma_d^2)$.

Usage

```
prepare_hmnp_data(
  data,
  id_col,
  alt_col,
  choice_col,
  covariate_cols,
  person_col = NULL,
  alt_covariate_cols = NULL,
  outside_opt_label = NULL,
  cf_residual_col = NULL,
  include_outside_option = TRUE
)
```

Arguments

`data`	Data frame containing choice data.
`id_col`	Name of the column identifying choice situations (tasks). Task ids only need to be unique within a respondent.
`alt_col`	Name of the column identifying alternatives.
`choice_col`	Name of the column indicating the chosen alternative (1 = chosen, 0 = not chosen).
`covariate_cols`	Vector of names of structural covariate columns (the random-coefficient dimensions).

`person_col` Name of the respondent column grouping choice situations. `NULL` (default) makes each choice situation its own respondent.

`alt_covariate_cols` Names of alternative-level covariate columns (constant within each alternative) forming the δ mean function. `NULL` (default) gives an intercept-only design (P = 1).

`outside_opt_label` Label of physical outside-option rows, removed when `include_outside_option = TRUE` (the outside good is implicit).

`cf_residual_col` Name of a first-stage residual column (control function for an endogenous covariate), appended to `X`. Default `NULL`.

`include_outside_option` Logical; if `TRUE` (default) an implicit outside option with systematic utility 0 is part of every choice set.

Details

The returned structure is identical to `prepare_hmnl_data()` (both preps share one internal engine), except there is no `rc_dist` field. Unlike `prepare_mnp_data()`, utilities are NOT differenced against a base alternative: the iid-shock model works in un-differenced utility space, so unbalanced choice sets are supported and the outside option is implicit (its latent utility is a stochastic $N(0, \sigma^2)$ draw in the kernel, systematic utility 0).

Value

A list of class `c("choicer_data_hmnp", "list")` with the same components as `prepare_hmnl_data()` (minus `rc_dist`).

See Also

`prepare_hmnl_data()` for the component-by-component description.

Examples

```
library(data.table)
set.seed(42)
N <- 20; T <- 3; J <- 4
dt <- data.table(
  pid  = rep(1:N, each = T * J),
  task = rep(seq_len(N * T), each = J),
  alt  = rep(1:J, N * T)
)
dt[, `:=`(x1 = rnorm(.N), x2 = runif(.N, -1, 1))]
dt[, choice := 0L]
dt[, choice := if (runif(1) < 0.8) sample(c(1L, rep(0L, J - 1))) else 0L,
   by = task]
input <- prepare_hmnp_data(dt, "task", "alt", "choice", c("x1", "x2"),
                           person_col = "pid")
input$Ti[1:5]
```

```
input$alt_mapping
```

`prepare_mnl_data` *Prepare inputs for multinomial logit estimation*

Description

Prepares and validates inputs for multinomial logit estimation routine.

Usage

```
prepare_mnl_data(
  data,
  id_col,
  alt_col,
  choice_col,
  covariate_cols,
  weights = NULL,
  outside_opt_label = NULL,
  include_outside_option = FALSE,
  weights_col = NULL,
  cluster_col = NULL
)
```

Arguments

`data`	Data frame containing choice data.
`id_col`	Name of the column identifying choice situations (individuals).
`alt_col`	Name of the column identifying alternatives.
`choice_col`	Name of the column indicating chosen alternative (1 = chosen, 0 = not chosen).
`covariate_cols`	Vector of names of columns to be used as covariates.
`weights`	Optional vector of weights for each choice situation. If `NULL`, equal weights are used. All weights must be finite and strictly positive.
`outside_opt_label`	Label for the outside option (if any). If `NULL`, no outside option is assumed.
`include_outside_option`	Logical indicating whether to include an outside option in the model.
`weights_col`	Optional name of a column in `data` holding per-row weights. The column must be constant within each `id_col` (one weight per choice situation) and is collapsed accordingly. Mutually exclusive with `weights`. All weights must be finite and strictly positive.
`cluster_col`	Optional name of a column in `data` holding cluster labels for cluster-robust standard errors (e.g. a person id when the same decision maker contributes several choice situations). Must be constant within each `id_col`. Collapsed to one label per choice situation and returned as `cluster`; used by `se_method = "cluster"` and `vcov(fit, type = "cluster")`.

Value

A list containing:

- `X`: Design matrix (sum(M) x K).
- `alt_idx`: Integer vector of alternative indices.
- `choice_idx`: Integer vector of chosen alternative indices.
- `M`: Integer vector with number of alternatives per choice situation.
- `N`: Number of choice situations.
- `weights`: Vector of weights.
- `cluster`: Vector of cluster labels (or `NULL`).
- `situation_ids`: Choice-situation ids in prepared (sorted) order.
- `include_outside_option`: Logical flag.
- `alt_mapping`: Data.table mapping alternatives to summary statistics.
- `dropped_cols`: Names of columns dropped due to collinearity, if any.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
input <- prepare_mnl_data(dt, "id", "alt", "choice", c("x1", "x2"))
str(input$X)
input$alt_mapping
```

---

prepare_mnp_data *Prepare inputs for Bayesian multinomial probit estimation*

Description

Prepares and validates inputs for Bayesian multinomial probit estimation. Covariates are differenced against the base alternative, so the design matrix has one row per (choice situation, non-base alternative) pair. Balanced choice sets are required: every choice situation must contain the same J alternatives.

Usage

```
prepare_mnp_data(
  data,
  id_col,
  alt_col,
  choice_col,
```

```
  covariate_cols,
  base_alt = NULL,
  use_asc = TRUE
)
```

Arguments

data	Data frame containing choice data.
id_col	Name of the column identifying choice situations (individuals).
alt_col	Name of the column identifying alternatives.
choice_col	Name of the column indicating chosen alternative (1 = chosen, 0 = not chosen).
covariate_cols	Vector of names of columns to be used as covariates.
base_alt	Label of the base (reference) alternative used for utility differencing. If NULL (default), the first alternative in sort order is used.
use_asc	Logical indicating whether to include alternative-specific constants (one intercept per non-base alternative).

Value

A list containing:

- X: Stacked differenced design matrix ((N * p) x K), covariate columns first, then ASC columns when use_asc = TRUE.
- y: Integer vector of choices (0 = base alternative, j in 1..p for the j-th non-base alternative), one per choice situation.
- p: Number of utility differences (J - 1).
- J: Number of alternatives.
- N: Number of choice situations.
- K: Number of columns of X.
- alt_mapping: Data.table mapping alternatives to summary statistics (the base alternative is alt_int = 1).
- base_alt: Resolved label of the base alternative.
- param_map: Named list of integer index vectors (beta, asc).
- use_asc: Logical flag.
- dropped_cols: Names of columns dropped due to collinearity, if any.
- data_spec: List with column name metadata.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
```

```
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
input <- prepare_mnp_data(dt, "id", "alt", "choice", c("x1", "x2"))
str(input$X)
input$alt_mapping
```

`prepare_mxl_data` *Prepare inputs for mixed logit estimation*

Description

Prepares and validates inputs for mixed logit estimation routine.

Usage

```
prepare_mxl_data(
  data,
  id_col,
  alt_col,
  choice_col,
  covariate_cols,
  random_var_cols,
  weights = NULL,
  outside_opt_label = NULL,
  include_outside_option = FALSE,
  rc_correlation = FALSE,
  weights_col = NULL,
  cluster_col = NULL
)
```

Arguments

`data`	Data frame containing choice data
`id_col`	Name of the column identifying choice situations (individuals)
`alt_col`	Name of the column identifying alternatives
`choice_col`	Name of the column indicating chosen alternative (1 = chosen, 0 = not chosen)
`covariate_cols`	Vector of names of columns to be used as covariates
`random_var_cols`	Vector of names of columns to be used as random variables
`weights`	Optional vector of weights for each choice situation. If NULL, equal weights are used. All weights must be finite and strictly positive.
`outside_opt_label`	Label for the outside option (if any). If NULL, no outside option is assumed.
`include_outside_option`	Logical indicating whether to include an outside option in the model.
`rc_correlation`	Logical indicating whether random coefficients are correlated. Default is FALSE.

`weights_col` Optional name of a column in `data` holding a per-row weight (constant within each choice situation, finite and strictly positive). Mutually exclusive with `weights`.

`cluster_col` Optional name of a column in `data` holding cluster labels for cluster-robust standard errors. Must be constant within each `id_col`; collapsed to one label per choice situation and returned as `cluster`.

Value

A `choicer_data_mxl` object (list) containing:

- `X`: Fixed-coefficient design matrix (sum(M) x K_x).
- `W`: Random-coefficient design matrix (sum(M) x K_w).
- `alt_idx`: Integer vector of alternative indices.
- `choice_idx`: Integer vector of chosen alternative indices.
- `M`: Integer vector with number of alternatives per choice situation.
- `N`: Number of choice situations.
- `weights`: Vector of weights.
- `cluster`: Vector of cluster labels (or `NULL`).
- `situation_ids`: Choice-situation ids in prepared (sorted) order.
- `include_outside_option`: Logical flag.
- `rc_correlation`: Logical flag.
- `alt_mapping`: data.table mapping alternatives to summary statistics.
- `dropped_cols`: Names of columns dropped due to collinearity, if any.
- `data_spec`: List with column-name metadata.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), w1 = rnorm(.N), w2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
input <- prepare_mxl_data(dt, "id", "alt", "choice", "x1", c("w1", "w2"))
str(input$X)
str(input$W)
```

prepare_nl_data *Prepare inputs for nested logit estimation*

Description

Validates inputs, builds design matrices, and constructs nest structure for nested logit estimation. Calls prepare_mnl_data internally for base data preparation, then adds nest-specific fields.

Usage

```
prepare_nl_data(
  data,
  id_col,
  alt_col,
  choice_col,
  covariate_cols,
  nest_col,
  weights = NULL,
  outside_opt_label = NULL,
  include_outside_option = FALSE,
  weights_col = NULL,
  cluster_col = NULL
)
```

Arguments

data	Data frame containing choice data.
id_col	Name of the column identifying choice situations (individuals).
alt_col	Name of the column identifying alternatives.
choice_col	Name of the column indicating chosen alternative (1 = chosen, 0 = not chosen).
covariate_cols	Vector of names of columns to be used as covariates.
nest_col	Name of the column mapping each alternative to its nest. Every alternative must belong to exactly one nest.
weights	Optional vector of weights for each choice situation. If NULL, equal weights are used. All weights must be finite and strictly positive.
outside_opt_label	Label for the outside option (if any). If NULL, no outside option is assumed.
include_outside_option	Logical indicating whether to include an outside option in the model.
weights_col	Optional name of a column in data holding per-row weights. The column must be constant within each id_col (one weight per choice situation) and is collapsed accordingly. Mutually exclusive with weights. All weights must be finite and strictly positive.
cluster_col	Optional name of a column in data holding cluster labels for cluster-robust standard errors. Must be constant within each id_col; collapsed to one label per choice situation and returned as cluster.

Value

A choicer_data_nl object (list) containing:

- All fields from prepare_mnl_data (X, alt_idx, choice_idx, M, N, weights, cluster, situation_ids, include_outside_option, alt_mapping, dropped_cols).
- nest_idx: Integer vector of length J mapping each alternative (in alt_mapping row order) to its nest.
- data_spec: List with column name metadata including nest_col.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 4
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, nest := ifelse(alt <= 2, "A", "B")]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
input <- prepare_nl_data(dt, "id", "alt", "choice", c("x1", "x2"), "nest")
input$nest_idx
input$alt_mapping
```

---

print.choicer_cs *Print a consumer surplus summary*

Description

Print a consumer surplus summary

Usage

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

Arguments

x	A choicer_cs object.
digits	Number of significant digits to print.
...	Additional arguments (ignored).

Value

The object invisibly.

print.choicer_fit *Print a choicer_fit object*

Description

Prints a brief summary of the fitted model.

Usage

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

Arguments

x	A choicer_fit object.
...	Additional arguments (ignored).

Value

The object invisibly.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
print(fit)
```

print.choicer_gof *Print goodness-of-fit measures*

Description

Print goodness-of-fit measures

Usage

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

Arguments

x	A choicer_gof object.
...	Additional arguments (ignored).

Value

The object invisibly.

---

print.choicer_hb *Print a hierarchical Bayes fit*

Description

Print a hierarchical Bayes fit

Usage

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

Arguments

x	A choicer_hmnl or choicer_hmnp object.
...	Additional arguments (ignored).

Value

The object invisibly.

Examples

```
sim <- simulate_hmnl_data(N = 50, T = 2, J = 3, seed = 42)
fit <- suppressWarnings(run_hmnlogit(sim$data, "task", "alt", "choice", c("x1", "x2"),
                   person_col = "pid",
                   mcmc = list(R = 300, burn = 100)))
print(fit)
```

print.choicer_mnp *Print a choicer_mnp object*

Description

Prints a brief summary of the fitted Bayesian multinomial probit model.

Usage

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

Arguments

x	A choicer_mnp object.
...	Additional arguments (ignored).

Value

The object invisibly.

Examples

```
library(data.table)
set.seed(42)
N <- 100; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnprobit(dt, "id", "alt", "choice", c("x1", "x2"),
                    mcmc = list(R = 300, burn = 100))
print(fit)
```

print.choicer_wtp *Print a WTP table*

Description

Print a WTP table

Usage

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

Arguments

x	A choicer_wtp object.
digits	Number of significant digits to print.
...	Additional arguments passed to print.data.frame.

Value

The object invisibly.

---

print.summary.choicer_hb

Print the summary of a hierarchical Bayes fit

Description

Print the summary of a hierarchical Bayes fit

Usage

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

Arguments

x	A summary.choicer_hb object.
...	Additional arguments (ignored).

Value

The object invisibly.

---

print.summary.choicer_mnl

Print summary for multinomial logit model

Description

Print summary for multinomial logit model

Usage

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

Arguments

x	A summary.choicer_mnl object.
...	Additional arguments (ignored).

Value

The object invisibly.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
print(summary(fit))
```

---

print.summary.choicer_mnp

Print summary for Bayesian multinomial probit model

Description

Print summary for Bayesian multinomial probit model

Usage

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

Arguments

x	A summary.choicer_mnp object.
...	Additional arguments (ignored).

Value

The object invisibly.

Examples

```
library(data.table)
set.seed(42)
N <- 100; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnprobit(dt, "id", "alt", "choice", c("x1", "x2"),
                    mcmc = list(R = 300, burn = 100))
print(summary(fit))
```

---

print.summary.choicer_mxl

Print summary for mixed logit model

Description

Print summary for mixed logit model

Usage

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

Arguments

x	A summary.choicer_mxl object.
...	Additional arguments (ignored).

Value

The object invisibly.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), w1 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mxlogit(
  data = dt, id_col = "id", alt_col = "alt", choice_col = "choice",
  covariate_cols = "x1", random_var_cols = "w1", S = 50L
)
print(summary(fit))
```

`print.summary.choicer_nl` *Print summary for nested logit model*

Description

Print summary for nested logit model

Usage

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

Arguments

`x`	A summary.choicer_nl object.
`...`	Additional arguments (ignored).

Value

The object invisibly.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 4
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, nest := ifelse(alt <= 2, "A", "B")]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_nestlogit(
  data = dt, id_col = "id", alt_col = "alt", choice_col = "choice",
  covariate_cols = c("x1", "x2"), nest_col = "nest"
)
print(summary(fit))
```

`recovery_table` *Parameter recovery table*

Description

Compares fitted coefficients to a set of true parameter values on the same scale as the estimator's internal parameterization. Returns one row per estimated parameter with true value, estimate, standard error, bias, relative bias (%), z-score against the truth, Wald CI, and a coverage indicator.

Usage

```
recovery_table(object, truth = NULL, level = 0.95, ...)

## S3 method for class 'choicer_fit'
recovery_table(object, truth = NULL, level = 0.95, ...)

## S3 method for class 'choicer_mnp'
recovery_table(object, truth = NULL, level = 0.95, ...)

## S3 method for class 'choicer_mc'
recovery_table(object, truth = NULL, level = 0.95, ...)

## S3 method for class 'choicer_hb'
recovery_table(object, truth = NULL, level = 0.95, ...)
```

Arguments

`object`	A `choicer_fit` object (MNL, MXL, or NL) or a `choicer_mc` result.
`truth`	Either a `choicer_sim` object (whose `$true_params` will be used) or a named list of true parameter values.
`level`	Confidence level for the Wald CI and coverage indicator. Default `0.95`.
`...`	Unused.

Details

For MXL fits the `sigma` block compares the raw Cholesky parameters (`L_params`), not the reconstructed covariance matrix. For log-normal random-coefficient means the raw `mu` estimate is compared directly; callers who want recovery on the DGP scale (`exp(mu)`) should transform both sides before calling.

When the estimator has normalized the first inside alternative's ASC to zero (which happens for MNL/MXL with `include_outside_option = FALSE` and no outside option baked into the fit), the first entry of `truth$delta` is dropped before the comparison so lengths match.

Value

See class-specific methods.

Methods (by class)

- `recovery_table(choicer_fit)`: Returns a `choicer_recovery` object (a `data.table`) with columns `parameter`, `group`, `true`, `estimate`, `se`, `bias`, `rel_bias_pct`, `z_vs_true`, `lower_ci`, `upper_ci`, `covers`.
- `recovery_table(choicer_mnp)`: Method for Bayesian MNP fits (`choicer_mnp`). The `estimate` column holds posterior means of the identified draws and `se` holds their posterior standard deviations, so `lower_ci` / `upper_ci` are normal-approximation credible intervals. In addition to the `beta` and `asc` blocks, a `sigma` block compares the identified covariance of the utility differences (lower triangle in the estimator's row-major `Sigma_ij` order); its first row is the `sigma_11 = 1` normalization and is exact by construction. `truth` must be on the identified scale, as returned by `simulate_mnp_data()`.
- `recovery_table(choicer_mc)`: For a `choicer_mc` object, delegates to `summary(object, level)` and returns a `choicer_mc_summary`. Inspect `object$replications` directly for per-rep detail.
- `recovery_table(choicer_hb)`: Method for hierarchical Bayes fits (`choicer_hmnl` / `choicer_hmnp`). `estimate` holds posterior means and `se` posterior standard deviations, so `lower_ci` / `upper_ci` are normal-approximation credible intervals. Blocks: `beta` (population means b vs `truth$beta`), `w` (diag(W) vs `diag(truth$W)`), `theta` (delta mean function), `sigma_d` (the SD, compared through the sqrt of the sigma_d^2 draws), and `delta` (per-alternative effects vs the realized `truth$delta`). For an HMNP fit, `truth` must be on the identified scale, as returned by `simulate_hmnp_data()`.

Examples

```
sim <- simulate_mnl_data(N = 2000, J = 4, seed = 123)
fit <- run_mnlogit(
  data = sim$data, id_col = "id", alt_col = "alt", choice_col = "choice",
  covariate_cols = c("x1", "x2"),
  outside_opt_label = 0L, include_outside_option = FALSE, use_asc = TRUE
)
recovery_table(fit, sim)
```

---

rhat *Split-$\widehat{R}$ convergence diagnostic*

Description

Computes the split-$\widehat{R}$ (potential scale reduction factor) of Gelman et al. for each column of a matrix of posterior draws. Every chain is split in half, so the diagnostic detects non-stationarity within a single chain as well as disagreement across chains; values near 1 indicate convergence, and values above roughly 1.05 warrant a longer run.

Usage

```
rhat(draws, rank = FALSE)
```

Arguments

draws	A matrix of posterior draws (rows = iterations, columns = parameters) for a single chain, or a list of such matrices (one per chain, identical dimensions).
rank	Logical; if FALSE (default) computes the classic split $\widehat{R}$ (unchanged from prior releases, bit-for-bit). If TRUE, computes the rank-normalized, folded $\widehat{R}$ of Vehtari, Gelman, Simpson, Carpenter & Buerkner (2021, *Bayesian Analysis*): the max of a bulk (rank-normalized draws) and a fold (rank- normalized absolute deviation from the pooled median) split-$R-hat$, which is more robust to heavy tails and scale differences across chains.

Value

Named numeric vector with one $\widehat{R}$ per parameter (NA for parameters with zero variance).

Examples

```
set.seed(42)
draws <- matrix(rnorm(2000), ncol = 2,
                dimnames = list(NULL, c("a", "b")))
rhat(draws)          # ~1: white noise is stationary
drifting <- cbind(a = cumsum(rnorm(1000)))
rhat(drifting)       # >> 1: a random walk is not
rhat(draws, rank = TRUE)   # rank-normalized variant
```

---

run_hmnlogit *Fit a hierarchical Bayesian multinomial logit (HMNL)*

Description

Runs the adaptive RW-Metropolis-within-Gibbs sampler for the hierarchical (random-coefficients, panel or cross-sectional) multinomial logit with a BLP-style alternative-level random effect:

$$U_{ijt} = x'_{ijt}\gamma_i + \delta_j + \epsilon_{ijt}, \qquad U_{iot} = \epsilon_{iot},$$

with i.i.d. Gumbel shocks (including on the implicit outside option, whose systematic utility is 0), $\beta_i \sim N(b, W)$ over the structural covariates ($\gamma_{ik} = \beta_{ik}$ or $\exp(\beta_{ik})$ per rc_dist), and $\delta_j = z'_j\theta + \xi_j$, $\xi_j \sim N(0, \sigma_d^2)$. Partial pooling shrinks each δ_j toward its characteristics-based mean $z'_j\theta$; the outside option anchors the level of δ (mean utility relative to the outside good), so no base alternative or sum-to-zero constraint is needed.

Usage

```
run_hmnlogit(
  data = NULL,
  id_col = NULL,
  alt_col = NULL,
  choice_col = NULL,
  covariate_cols = NULL,
```

```
  person_col = NULL,
  alt_covariate_cols = NULL,
  outside_opt_label = NULL,
  cf_residual_col = NULL,
  input_data = NULL,
  include_outside_option = TRUE,
  rc_dist = NULL,
  prior = list(),
  mcmc = list(),
  chains = 1,
  keep_beta_i = c("means", "draws", "none"),
  keep_data = TRUE
)
```

Arguments

`data`	Data frame (convenience pathway). Supply either `data` (with the column names) or `input_data`, not both.
`id_col`	Name of the column identifying choice situations (tasks). Task ids only need to be unique within a respondent.
`alt_col`	Name of the column identifying alternatives.
`choice_col`	Name of the column indicating the chosen alternative (1 = chosen, 0 = not chosen).
`covariate_cols`	Vector of names of structural covariate columns (the random-coefficient dimensions).
`person_col`	Name of the respondent column grouping choice situations. `NULL` (default) makes each choice situation its own respondent.
`alt_covariate_cols`	Names of alternative-level covariate columns (constant within each alternative) forming the δ mean function. `NULL` (default) gives an intercept-only design (P = 1).
`outside_opt_label`	Label of physical outside-option rows, removed when `include_outside_option = TRUE` (the outside good is implicit).
`cf_residual_col`	Name of a first-stage residual column (control function for an endogenous covariate), appended to `X`. Default `NULL`.
`input_data`	A `choicer_data_hmnl` object from `prepare_hmnl_data()` (advanced pathway).
`include_outside_option`	Logical; if `TRUE` (default) an implicit outside option with systematic utility 0 is part of every choice set.
`rc_dist`	Integer vector, one entry per column of `covariate_cols`: `0` for a normal random coefficient, `1` for log-normal (the coefficient enters utility as `exp(beta_ik)`; hierarchy normal on the log scale). Default `NULL` is all-normal. Automatically aligned through dropped columns; a `cf_residual_col` coordinate is always normal.

prior	Named list overriding prior defaults: b_bar (0), A (0.01 I), nu (K + 3), V (nu I), theta_bar (0), A_theta (0.01 I), sd_prior (list(half_cauchy = TRUE, s_d = 1, c0 = 3, d0 = 3)).
mcmc	Named list overriding MCMC defaults: R (10000), burn (R %/% 5), thin (1), seed (drawn via sample.int() so set.seed() governs), trace (0), s_init (2.38 / sqrt(K)), accept_target (0.234).
chains	Number of independent chains (seeds offset by 1, run sequentially). Chain 1 provides the reported draws; all chains feed the rank-normalized split-R-hat table and the retained chains field (all per-chain b/w_vech/delta/theta/sigma_d2/loglik draws, consumed by ess(), mcse(), and traceplot()).
keep_beta_i	"means" (default) stores posterior means/SDs of the individual-level β_i; "draws" additionally stores the full (K, N, R_keep) draw cube (memory-guarded, budgeted per chain); "none" stores neither.
keep_data	Logical; keep the prepared data on the fit (default TRUE, needed by post-estimation methods).

Details

Initialization. β_i start at the pooled MNL maximum likelihood estimate over the structural covariates (log-normal coordinates transformed to the chain scale with a warn-and-clamp at 0.05); δ starts at shrunk log choice-share contrasts against the outside option; θ at the OLS regression of the initial δ on Z.

Priors. $b \sim N(b_bar, A^{-1})$, $W \sim IW(\nu, V)$, $\theta \sim N(\theta_bar, A_\theta^{-1})$, and $\sigma_d \sim$ half-Cauchy$(0, s_d)$ via the Makalic-Schmidt scale mixture (set sd_prior$half_cauchy = FALSE for a plain $IG(c_0, d_0)$ on σ_d^2).

Endogeneity. If a price-like covariate is endogenous (correlated with ξ_j), supply a first-stage residual via cf_residual_col (Petrin & Train 2010); posterior uncertainty does NOT propagate first-stage estimation error.

For beta_i draws at very large scale beyond the memory guard's threshold, a future disk-streaming path (writing each kept slice to disk instead of retaining it in memory) is on the roadmap but not built in this phase; users needing per-respondent draws at that scale should reduce R, reduce chains, or use keep_beta_i = "means".

Value

A choicer_hmnl object (classed c("choicer_hmnl", "choicer_hb")) with posterior summaries (coefficients, se, vcov for b; theta_summary; sigma_d2_summary; W_mean; delta and xi quality-ladder tables; beta_i), the raw thinned draws (chain 1), acceptance diagnostics in accept, the rank-normalized split-R-hat table in rhat, all chains' retained draws in chains, and sampler metadata.

See Also

prepare_hmnl_data(), simulate_hmnl_data(), recovery_table(), rhat(), ess(), mcse(), traceplot()

Examples

```
sim <- simulate_hmnl_data(N = 100, T = 3, J = 4, seed = 42)
fit <- suppressWarnings(run_hmnlogit(sim$data, "task", "alt", "choice", c("x1", "x2"),
                    person_col = "pid", alt_covariate_cols = "z1",
                    mcmc = list(R = 500, burn = 200)))
summary(fit)
coef(fit, component = "delta")
```

`run_hmnprobit` *Fit a hierarchical Bayesian multinomial probit (HMNP)*

Description

Runs the fully conjugate Albert-Chib Gibbs sampler for the hierarchical multinomial probit with iid normal utility shocks in un-differenced utility space:

$$U_{ijt} = x'_{ijt}\beta_i + \delta_j + \epsilon_{ijt}, \qquad U_{iot} = \epsilon_{iot}, \qquad \epsilon \sim N(0, \sigma^2),$$

choice by argmax within the task including the stochastic implicit outside option, $\beta_i \sim N(b, W)$ (normal coordinates only — log-normal would break conjugacy), and $\delta_j = z'_j\theta + \xi_j, \xi_j \sim N(0, \sigma_d^2)$.

Usage

```
run_hmnprobit(
  data = NULL,
  id_col = NULL,
  alt_col = NULL,
  choice_col = NULL,
  covariate_cols = NULL,
  person_col = NULL,
  alt_covariate_cols = NULL,
  outside_opt_label = NULL,
  cf_residual_col = NULL,
  input_data = NULL,
  include_outside_option = TRUE,
  prior = list(),
  mcmc = list(),
  chains = 1,
  keep_beta_i = c("means", "draws", "none"),
  keep_data = TRUE
)
```

Arguments

`data`	Data frame (convenience pathway). Supply either `data` (with the column names) or `input_data`, not both.

`id_col` Name of the column identifying choice situations (tasks). Task ids only need to be unique within a respondent.

`alt_col` Name of the column identifying alternatives.

`choice_col` Name of the column indicating the chosen alternative (1 = chosen, 0 = not chosen).

`covariate_cols` Vector of names of structural covariate columns (the random-coefficient dimensions).

`person_col` Name of the respondent column grouping choice situations. `NULL` (default) makes each choice situation its own respondent.

`alt_covariate_cols` Names of alternative-level covariate columns (constant within each alternative) forming the δ mean function. `NULL` (default) gives an intercept-only design (P = 1).

`outside_opt_label` Label of physical outside-option rows, removed when `include_outside_option = TRUE` (the outside good is implicit).

`cf_residual_col` Name of a first-stage residual column (control function for an endogenous covariate), appended to `X`. Default `NULL`.

`input_data` A `choicer_data_hmnp` object from `prepare_hmnp_data()`.

`include_outside_option` Logical; if `TRUE` (default) an implicit outside option with systematic utility 0 is part of every choice set.

`prior` As in `run_hmnlogit()`, plus `a0` (3) and `s0` (3), the inverse-gamma shape/scale on the non-identified σ^2.

`mcmc` Named list overriding MCMC defaults: `R` (10000), `burn` (R %/% 5), `thin` (1), `seed` (drawn via `sample.int()` so `set.seed()` governs), `trace` (0). No proposal-scale settings: the sampler is fully conjugate, with no Metropolis steps.

`chains` Number of independent chains (seeds offset by 1, run sequentially). Chain 1 provides the reported draws; all chains feed the rank-normalized split-R-hat table and the retained `chains` field (all per-chain `b`/`w_vech`/`delta`/`theta`/`sigma_d2`/`loglik` draws, consumed by `ess()`, `mcse()`, and `traceplot()`).

`keep_beta_i` `"means"` (default) stores posterior means/SDs of the individual-level β_i; `"draws"` additionally stores the full (K, N, R_keep) draw cube (memory-guarded, budgeted per chain); `"none"` stores neither.

`keep_data` Logical; keep the prepared data on the fit (default `TRUE`, needed by post-estimation methods).

Details

Identification. The probit likelihood is invariant to a common rescaling of utilities and σ, so the chain runs on the non-identified parameterization (free σ^2, better mixing via parameter expansion) and every kept draw is normalized by the matching power of the CURRENT σ: reported b/σ, W/σ^2, δ/σ, θ/σ, σ_d^2/σ^2. Raw chains are kept in `draws$*_raw`. The outside option anchors the location of δ exactly as in `run_hmnlogit()`.

For `beta_i` draws at very large scale beyond the memory guard's threshold, a future disk-streaming path (writing each kept slice to disk instead of retaining it in memory) is on the roadmap but not built in this phase; users needing per-respondent draws at that scale should reduce `R`, reduce `chains`, or use `keep_beta_i = "means"`.

Value

A `choicer_hmnp` object (classed `c("choicer_hmnp", "choicer_hb")`); the same layout as `run_hmnlogit()`'s return, with all reported summaries on the identified scale, raw chains in `draws$*_raw`, the non-identified `draws$sigma2` trace, and all chains' retained draws (identified scale) in `chains`.

See Also

`prepare_hmnp_data()`, `simulate_hmnp_data()`, `run_hmnlogit()`, `ess()`, `mcse()`, `traceplot()`

Examples

```
sim <- simulate_hmnp_data(N = 100, T = 3, J = 4, seed = 42)
fit <- suppressWarnings(run_hmnprobit(sim$data, "task", "alt", "choice", c("x1", "x2"),
                    person_col = "pid", alt_covariate_cols = "z1",
                    mcmc = list(R = 500, burn = 200)))
summary(fit)
```

`run_mnlogit` *Runs multinomial logit estimation*

Description

Estimates a multinomial logit model via maximum likelihood.

Usage

```
run_mnlogit(
  data = NULL,
  id_col = NULL,
  alt_col = NULL,
  choice_col = NULL,
  covariate_cols = NULL,
  input_data = NULL,
  optimizer = NULL,
  control = list(),
  weights = NULL,
  weights_col = NULL,
  outside_opt_label = NULL,
  include_outside_option = FALSE,
  use_asc = TRUE,
  keep_data = TRUE,
```

```
  scale_vars = c("none", "sd", "mad", "iqr"),
  se_method = c("hessian", "bhhh", "sandwich", "cluster"),
  cluster_col = NULL,
  nloptr_opts = NULL
)
```

Arguments

data	Data frame containing choice data (convenience workflow). Mutually exclusive with input_data.
id_col	Name of the column identifying choice situations (individuals).
alt_col	Name of the column identifying alternatives.
choice_col	Name of the column indicating chosen alternative (1 = chosen, 0 = not chosen).
covariate_cols	Vector of names of columns to be used as covariates.
input_data	List output from prepare_mnl_data (advanced workflow). Mutually exclusive with data.
optimizer	Optimizer to use: "nloptr" (default), "optim", or a custom function with signature f(theta_init, eval_f, lower, upper, control) where eval_f(theta) returns list(objective, gradient). Must return a list with par/value (or solution/objective). If the custom function accepts control or ..., the control argument is forwarded; otherwise it is silently ignored.
control	List of optimizer-specific control parameters passed to the chosen optimizer (e.g., list(maxeval = 2000) for nloptr).
weights	Optional vector of weights for each choice situation. If NULL, equal weights are used. All weights must be finite and strictly positive.
weights_col	Optional name of a column in data holding per-row weights (convenience workflow only). The column must be constant within each id_col (one weight per choice situation) and is collapsed accordingly. Mutually exclusive with weights. All weights must be finite and strictly positive. Used for choice-based / WESML weighting; pair with se_method = "sandwich" for valid inference.
outside_opt_label	Label for the outside option (if any). If NULL, no outside option is assumed.
include_outside_option	Logical indicating whether to include an outside option in the model.
use_asc	Logical indicating whether to include alternative-specific constants (ASCs) in the model.
keep_data	Logical. If TRUE (default), stores prepared data in the returned object for predict() and post-estimation functions.
scale_vars	Pre-estimation column scaling for the design matrix. One of "none" (default), "sd" (sample standard deviation), "mad" (stats::mad), or "iqr" (stats::IQR(x) / 1.349). When not "none", every column of X is divided by the chosen scale before optimization to improve Hessian conditioning. Coefficients and standard errors are back-transformed to the user's natural units via the delta method, so reported quantities are invariant to this choice.

`se_method` Method for computing standard errors: `"hessian"` (default, analytical Hessian), `"bhhh"` (outer product of gradients), `"sandwich"` (robust Huber–White / WESML variance $A^{-1}BA^{-1}$), or `"cluster"` (cluster-robust sandwich; requires `cluster_col` or a prepared `input_data` with a `cluster` field). Use `"sandwich"` under choice-based / WESML weighting. Any of these can also be recomputed post hoc via `vcov(fit, type = )`.

`cluster_col` Optional name of a column in `data` holding cluster labels for cluster-robust standard errors (e.g. a person id when the same decision maker contributes several choice situations). Must be constant within each `id_col`. Supplying `cluster_col` without an explicit `se_method` selects `se_method = "cluster"`.

`nloptr_opts` Deprecated. Use `optimizer` and `control` instead.

Details

Two workflows are supported:

Convenience (default) Supply `data` and column names. Data preparation (`prepare_mnl_data`) is handled automatically.

Advanced Call `prepare_mnl_data` yourself and pass the result via `input_data`.

Value

A `choicer_mnl` object (inherits from `choicer_fit`). Standard S3 methods available: `summary()`, `coef()`, `vcov()`, `logLik()`, `AIC()`, `BIC()`, `nobs()`, `predict()`.

Examples

```
library(data.table)
set.seed(42)
N <- 100; J <- 3; beta_true <- c(1.0, -0.5)
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, V := drop(as.matrix(.SD) %*% beta_true), .SDcols = c("x1","x2")]
dt[, prob := exp(V) / sum(exp(V)), by = id]
dt[, choice := as.integer(alt == sample(alt, 1, prob = prob)), by = id]

fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
summary(fit)
coef(fit)
AIC(fit)
predict(fit, type = "shares")
```

run_mnprobit *Runs Bayesian multinomial probit estimation*

Description

Estimates a multinomial probit model by Gibbs sampling with data augmentation (Albert & Chib 1993; McCulloch & Rossi 1994). The model is specified in utility differences against a base alternative: for choice situation i with J alternatives, $w_i = X_i\beta + \epsilon_i$ with $\epsilon_i \sim N_{J-1}(0, \Sigma)$.

Usage

```
run_mnprobit(
  data = NULL,
  id_col = NULL,
  alt_col = NULL,
  choice_col = NULL,
  covariate_cols = NULL,
  input_data = NULL,
  base_alt = NULL,
  use_asc = TRUE,
  prior = list(),
  mcmc = list(),
  keep_data = TRUE
)
```

Arguments

`data`	Data frame containing choice data (convenience workflow). Mutually exclusive with `input_data`.
`id_col`	Name of the column identifying choice situations (individuals).
`alt_col`	Name of the column identifying alternatives.
`choice_col`	Name of the column indicating chosen alternative (1 = chosen, 0 = not chosen).
`covariate_cols`	Vector of names of columns to be used as covariates.
`input_data`	List output from `prepare_mnp_data` (advanced workflow). Mutually exclusive with `data`.
`base_alt`	Label of the base (reference) alternative used for utility differencing. If `NULL` (default), the first alternative in sort order is used.
`use_asc`	Logical indicating whether to include alternative-specific constants (one intercept per non-base alternative in the differenced utilities).
`prior`	Named list of prior settings, merged over defaults:
	`beta_bar` Prior mean of β (default `rep(0, K)`).
	`A` Prior precision of β (default `0.01 * diag(K)`).
	`nu` Inverse-Wishart degrees of freedom (default `p + 3`).
	`V` Inverse-Wishart scale matrix (default `nu * diag(p)`).

mcmc	Named list of MCMC settings, merged over defaults: R Total Gibbs iterations (default 10000). burn Burn-in iterations discarded (default floor(R / 5)). thin Keep every thin-th post-burn-in draw (default 1). seed Master RNG seed (default: drawn from R's RNG). trace Print progress every trace iterations (default 0, silent).
keep_data	Logical. If TRUE (default), stores prepared data in the returned object.

Details

Two workflows are supported:

Convenience (default) Supply data and column names. Data preparation (prepare_mnp_data) is handled automatically.

Advanced Call prepare_mnp_data yourself and pass the result via input_data.

Identification. The multinomial probit likelihood is invariant to a common rescaling $(\beta, \Sigma) \to (c\beta, c^2\Sigma)$. The sampler runs on the non-identified parameterization (unrestricted Σ with an inverse-Wishart prior) and identified quantities are computed by normalizing each kept draw by σ_{11}: $\beta/\sqrt{\sigma_{11}}$ and Σ/σ_{11}. This is the McCulloch & Rossi (1994) default, which keeps all Gibbs conditionals conjugate and mixes better than the fully identified sampler of McCulloch, Polson & Rossi (2000). Reported coefficients, standard deviations, and credible intervals are posterior summaries of the identified draws.

Reproducibility. The sampler uses its own thread-safe RNG with one stream per (iteration, observation), so results are reproducible independent of the number of OpenMP threads (see set_num_threads()). When mcmc$seed is not supplied, a master seed is drawn from R's RNG, so set.seed() controls the run.

Scope. Balanced choice sets are required: every choice situation must contain the same J alternatives. To model an outside option, include it as explicit rows with zero covariates and set base_alt to its label.

Value

A choicer_mnp object. S3 methods available: summary(), coef() (posterior means of identified coefficients), vcov() (posterior covariance of identified coefficient draws), nobs(). Posterior draws are stored in $draws (beta / sigma on the identified scale, beta_raw / sigma_raw unnormalized).

References

Albert, J. H., & Chib, S. (1993). Bayesian Analysis of Binary and Polychotomous Response Data. *Journal of the American Statistical Association*, 88(422), 669-679.

McCulloch, R., & Rossi, P. E. (1994). An exact likelihood analysis of the multinomial probit model. *Journal of Econometrics*, 64(1-2), 207-240.

Examples

```
library(data.table)
set.seed(42)
N <- 200; J <- 3; beta_true <- c(1.0, -0.5)
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, U := drop(as.matrix(.SD) %*% beta_true) + rnorm(.N), .SDcols = c("x1", "x2")]
dt[, choice := as.integer(U == max(U)), by = id]

fit <- run_mnprobit(dt, "id", "alt", "choice", c("x1", "x2"),
                    mcmc = list(R = 500, burn = 100))
summary(fit)
coef(fit)
```

`run_mxlogit` *Runs mixed logit estimation*

Description

Estimates a mixed logit model via simulated maximum likelihood.

Usage

```
run_mxlogit(
  data = NULL,
  id_col = NULL,
  alt_col = NULL,
  choice_col = NULL,
  covariate_cols = NULL,
  random_var_cols = NULL,
  input_data = NULL,
  eta_draws = NULL,
  S = 100L,
  rc_dist = NULL,
  rc_mean = FALSE,
  rc_correlation = FALSE,
  use_asc = TRUE,
  theta_init = NULL,
  lower = NULL,
  upper = NULL,
  optimizer = NULL,
  control = list(),
  se_method = c("hessian", "bhhh", "sandwich", "cluster"),
  scale_vars = c("none", "sd", "mad", "iqr"),
  weights = NULL,
  outside_opt_label = NULL,
```

```
  include_outside_option = FALSE,
  draws = c("store", "generate"),
  seed = NULL,
  scramble = c("permuted", "none", "owen"),
  keep_data = TRUE,
  nloptr_opts = NULL,
  weights_col = NULL,
  cluster_col = NULL
)
```

Arguments

`data`	Data frame containing choice data (convenience workflow). Mutually exclusive with `input_data`.
`id_col`	Name of the column identifying choice situations.
`alt_col`	Name of the column identifying alternatives.
`choice_col`	Name of the column indicating chosen alternative (1/0).
`covariate_cols`	Vector of column names for fixed covariates.
`random_var_cols`	Vector of column names for random coefficients.
`input_data`	List output from `prepare_mxl_data` (advanced workflow). Mutually exclusive with `data`.
`eta_draws`	Array of shape K_w x S x N with standard normal draws. Required for the advanced workflow; auto-generated from `S` in the convenience workflow.
`S`	Integer number of Halton draws per individual (convenience workflow only). Default 100.
`rc_dist`	Integer vector indicating distribution of random coefficients (0 = normal, 1 = log-normal). Default: all normal.
`rc_mean`	Logical indicating whether to estimate means for random coefficients.
`rc_correlation`	Logical indicating whether random coefficients are correlated (convenience workflow). Ignored when `input_data` is used (taken from the prepared data).
`use_asc`	Logical indicating whether to include alternative-specific constants.
`theta_init`	Initial parameter vector in natural-scale units. If `NULL`, defaults to zeros for the β, μ, and ASC blocks, and `log(0.5)` on the Cholesky diagonal (so each diagonal factor $L_{pp} = 0.5$, i.e. a moderate random-coefficient variance of `0.25`). The zero-on-diagonal alternative corresponds to $L_{pp} = 1$ (unit RC variance), which often lets the first L-BFGS step overshoot.
`lower, upper`	Optional parameter bounds for the optimizer, in natural-scale units (forward-transformed internally to scaled space when `scale_vars != "none"`). Each accepts three forms: `NULL` (default) Unbounded (`-Inf`/`Inf`). **Unnamed numeric vector of length** `n_params` Full-length vector ordered exactly like `theta_init` (the nloptr-native form).

Named numeric vector Names must be a subset of the parameter names (β block: column names of `X`; μ block: `Mu_<col>` (if `rc_mean = TRUE`); Cholesky block: `L_<i><j>` for $i \geq j$; ASC block: `ASC_<level>`). Unlisted parameters default to $\pm\infty$. This is the recommended form for typical use, e.g. `lower = c(L_11 = -5, L_22 = -5)` to clip Cholesky diagonals.

`optimizer` Optimizer to use: `"nloptr"` (default), `"optim"`, or a custom function. See `run_mnlogit` for details.

`control` List of optimizer-specific control parameters.

`se_method` Method for computing standard errors. One of `"hessian"` (default) for the analytical Hessian of the simulated log-likelihood, `"bhhh"` for the BHHH/outer-product-of-gradients (OPG) estimator, `"sandwich"` for the robust (Huber-White) variance $V = A^{-1}BA^{-1}$ (bread A = weighted negated Hessian, meat B = weight-squared OPG), or `"cluster"` for the cluster-robust sandwich (requires `cluster_col` or a prepared `input_data` with a `cluster` field). Use `"sandwich"` for valid inference under choice-based / WESML weighting, where the inverse-Hessian and ordinary BHHH are invalid; it reduces to the usual robust variance under uniform weights. BHHH scales better to large problems (many alternatives or simulation draws) but may underestimate standard errors in finite samples or away from the optimum. Any of these can also be recomputed post hoc via `vcov(fit, type = )`. Note that clustering repairs the inference, not the estimand: the MXL likelihood treats each choice situation as an independent draw from the mixing distribution; for panel random coefficients use `run_hmnlogit`.

`scale_vars` Pre-estimation column scaling for design matrices. One of `"none"` (default), `"sd"` (sample standard deviation), `"mad"` (`stats::mad`, i.e. 1.4826 $\times$ median absolute deviation; SD-equivalent under normality), or `"iqr"` (`stats::IQR(x) / 1.349`; also SD-equivalent under normality). When not `"none"`, every column of `X` and `W` is divided by the chosen scale before optimization to improve Hessian conditioning. Robust scales (`"mad"`/`"iqr"`) better capture the bulk for heavy-tailed columns where SD is dominated by outliers, but `stats::mad` can return zero when more than half of a column's entries are identical (e.g., a sparse 0/1 dummy) and will then trigger the same near-constant-column error as `"sd"`. Coefficients and standard errors are back-transformed to the user's natural units via the delta method, so reported quantities are invariant to this choice. Columns of `W` associated with log-normal random coefficients (`rc_dist == 1`) are passed through unchanged, since the shifted log-normal parameterization does not admit a closed-form back-transform under multiplicative scaling.

`weights` Optional weight vector (convenience workflow). If `NULL`, equal weights are used. All weights must be finite and strictly positive.

`outside_opt_label`
Label for the outside option (convenience workflow).

`include_outside_option`
Logical whether to include an outside option (convenience workflow).

`draws` Draw storage mode. One of `"store"` (default) or `"generate"`. `"store"` pre-materializes the full $K_w \times S \times N$ Halton cube (existing behavior, exact reproducibility). `"generate"` computes each individual's draws on-the-fly in C++ from a stored seed, eliminating the O(N) cube; recommended for memory-constrained or large-N settings. With `scramble = "permuted"`, each base-b

digit position in each dimension receives a seeded permutation shared across sequence indices. This is not Owen's nested-uniform scramble and does not carry standard randomized-QMC unbiasedness or error-estimation guarantees. Only supported in the convenience workflow.

`seed` Integer master seed for the on-the-fly generator. Used only when `draws = "generate"`. If `NULL` (default), a seed is drawn from R's RNG at call time (so `set.seed()` governs reproducibility). Ignored when `draws = "store"`.

`scramble` Scrambling mode for on-the-fly Halton draws. One of `"permuted"` (default) for seeded position-wise digit permutations or `"none"` for plain Halton (identity permutations). The historical value `"owen"` is accepted with a deprecation warning as an alias for `"permuted"`; the implementation is not Owen's nested-uniform scramble. `"none"` reproduces the randtoolbox sequence exactly. Simulation-draw sensitivity should be assessed by increasing `S` and, for `"permuted"`, varying `seed`. Used only when `draws = "generate"`.

`keep_data` Logical. If `TRUE` (default), stores prepared data in the returned object for post-estimation functions.

`nloptr_opts` Deprecated. Use `optimizer` and `control` instead.

`weights_col` Optional name of a column in `data` holding a per-row weight (constant within each choice situation, finite and strictly positive). Mutually exclusive with `weights`; the recommended way to pass WESML weights from `sample_by_choice` / `wesml_weights`, since alignment is by id rather than by position. Convenience workflow only. If `data` carries choice-based-sampling provenance (a `"choice_sampling"` attribute, as attached by `sample_by_choice` / `wesml_weights`) and neither `weights` nor `weights_col` is supplied, the recorded weight column is auto-detected and applied (with a message); if that column is absent the call errors rather than silently fitting an unweighted model under a WESML label.

`cluster_col` Optional name of a column in `data` holding cluster labels for cluster-robust standard errors (e.g. a person id when the same decision maker contributes several choice situations). Must be constant within each `id_col`. Supplying `cluster_col` without an explicit `se_method` selects `se_method = "cluster"`.

Details

Two workflows are supported:

Convenience Supply `data` and column names. Data preparation (`prepare_mxl_data`) and Halton draw generation (`get_halton_normals`) are handled automatically.

Advanced Call `prepare_mxl_data` and `get_halton_normals` yourself, then pass the results via `input_data` and `eta_draws`.

Value

A `choicer_mxl` object (inherits from `choicer_fit`). Standard S3 methods available: `summary()`, `coef()`, `vcov()`, `logLik()`, `AIC()`, `BIC()`, `nobs()`.

Examples

```
library(data.table)
set.seed(42)
N <- 100; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), w1 = rnorm(.N), w2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]

fit <- run_mxlogit(
  data = dt, id_col = "id", alt_col = "alt", choice_col = "choice",
  covariate_cols = "x1", random_var_cols = c("w1", "w2"), S = 50L
)
summary(fit)
```

---

run_nestlogit *Runs nested logit estimation*

Description

Estimates a nested logit model via maximum likelihood.

Usage

```
run_nestlogit(
  data = NULL,
  id_col = NULL,
  alt_col = NULL,
  choice_col = NULL,
  covariate_cols = NULL,
  nest_col = NULL,
  input_data = NULL,
  use_asc = TRUE,
  theta_init = NULL,
  param_names = NULL,
  optimizer = NULL,
  control = list(),
  weights = NULL,
  weights_col = NULL,
  outside_opt_label = NULL,
  include_outside_option = FALSE,
  keep_data = TRUE,
  se_method = c("hessian", "numeric", "bhhh", "sandwich", "cluster"),
  cluster_col = NULL,
  nloptr_opts = NULL
)
```

Arguments

data	Data frame containing choice data (convenience workflow). Mutually exclusive with input_data.
id_col	Name of the column identifying choice situations.
alt_col	Name of the column identifying alternatives.
choice_col	Name of the column indicating chosen alternative (1/0).
covariate_cols	Vector of column names for covariates.
nest_col	Name of the column mapping each alternative to its nest (convenience workflow).
input_data	List containing prepared input data for estimation (advanced workflow). Mutually exclusive with data.
use_asc	Logical indicating whether to include alternative specific constants (ASCs).
theta_init	Optional initial parameter vector. If NULL, a default vector is used.
param_names	Optional vector of parameter names. If NULL, default names are generated.
optimizer	Optimizer to use: "nloptr" (default), "optim", or a custom function. See run_mnlogit for details.
control	List of optimizer-specific control parameters.
weights	Optional weight vector (convenience workflow). If NULL, equal weights are used. All weights must be finite and strictly positive.
weights_col	Optional name of a column in data holding per-row weights (convenience workflow only). The column must be constant within each id_col (one weight per choice situation) and is collapsed accordingly. Mutually exclusive with weights. All weights must be finite and strictly positive. Used for choice-based / WESML weighting; pair with se_method = "sandwich" for valid inference.
outside_opt_label	Label for the outside option (convenience workflow).
include_outside_option	Logical whether to include an outside option (convenience workflow).
keep_data	Logical. If TRUE (default), stores prepared data in the returned object for post-estimation functions.
se_method	Method for computing standard errors: "hessian" (default, analytical Hessian via nl_loglik_hessian_parallel), "numeric" (finite-difference oracle via nl_loglik_numeric_hessian), "bhhh" (outer product of gradients via nl_bhhh_parallel), "sandwich" (robust Huber–White / WESML variance $A^{-1}BA^{-1}$), or "cluster" (cluster-robust sandwich; requires cluster_col or a prepared input_data with a cluster field). Use "sandwich" under choice-based / WESML weighting. Any of these can also be recomputed post hoc via vcov(fit, type =).
cluster_col	Optional name of a column in data holding cluster labels for cluster-robust standard errors (e.g. a person id when the same decision maker contributes several choice situations). Must be constant within each id_col. Supplying cluster_col without an explicit se_method selects se_method = "cluster".
nloptr_opts	Deprecated. Use optimizer and control instead.

Details

Two workflows are supported:

Convenience Supply `data` and column names (including `nest_col`). Data preparation (`prepare_nl_data`) is handled automatically.

Advanced Call `prepare_nl_data` (or build the input list manually) and pass it via `input_data`.

Value

A `choicer_nl` object (inherits from `choicer_fit`). Standard S3 methods available: `summary()`, `coef()`, `vcov()`, `logLik()`, `AIC()`, `BIC()`, `nobs()`.

Examples

```
library(data.table)
set.seed(42)
N <- 100; J <- 4
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, nest := ifelse(alt <= 2, "A", "B")]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]

fit <- run_nestlogit(
  data = dt, id_col = "id", alt_col = "alt", choice_col = "choice",
  covariate_cols = c("x1", "x2"), nest_col = "nest"
)
summary(fit)
```

---

sample_by_choice *Draw a choice-based sample stratified by the chosen alternative*

Description

Subsamples whole choice situations from a population data set according to fixed per-stratum quotas, where strata are defined by the *chosen* alternative. The input data set is treated as the population, so the population shares $Q(j)$ are known exactly; the returned sample carries a ready-to-use WESML weight column (see `wesml_weights`).

Usage

```
sample_by_choice(
  data,
  id_col,
  alt_col,
  choice_col,
  n_per_alt = NULL,
```

```
  frac_per_alt = NULL,
  seed = NULL,
  weight_name = ".wesml_weight",
  outside_opt_label = NULL,
  include_outside_option = FALSE
)
```

Arguments

`data`, `id_col`, `alt_col`, `choice_col`	As in `wesml_weights`.
`n_per_alt`	Either a single integer applied to every stratum, or a named integer vector of per-stratum counts (names matched to `as.character(alt)`, covering all strata). Mutually exclusive with `frac_per_alt`.
`frac_per_alt`	Either a single fraction in `[0, 1]` applied to every stratum, or a named numeric vector of per-stratum fractions. Mutually exclusive with `n_per_alt`.
`seed`	Optional integer seed for reproducible sampling.
`weight_name`	Name of the attached weight column (default `".wesml_weight"`).
`outside_opt_label`, `include_outside_option`	As in `wesml_weights` (for an implicit outside good).

Details

Sampling is by choice situation (id), never by row: all alternative-rows of a sampled situation are kept together. Sampling is **without replacement**.

Value

A `data.table` subsample with the weight column appended and `"Q"`, `"H"`, and `"choice_sampling"` attributes (the last records the scheme, shares, quotas, and `meat = "robust"`).

References

Manski, C. F. and Lerman, S. R. (1977). *Econometrica* 45(8), 1977-1988.

See Also

`wesml_weights`, `run_mxlogit`

Examples

```
library(data.table)
set.seed(1)
N <- 600L; J <- 3L
pop <- data.table(id = rep(seq_len(N), each = J), alt = rep(1:J, N))
pop[, x1 := rnorm(.N)]
pop[, w1 := rnorm(.N)]
pop[, choice := as.integer(seq_len(.N) == sample.int(.N, 1L)), by = id]
```

```
s <- sample_by_choice(pop, "id", "alt", "choice", n_per_alt = 50L, seed = 1L)
attr(s, "choice_sampling")$H   # realized sample shares
head(s[[".wesml_weight"]])
```

`set_num_threads` *Set the number of OpenMP threads used by choicer*

Description

Set the number of OpenMP threads used by choicer

Usage

```
set_num_threads(n_threads)
```

Arguments

`n_threads`	Positive integer number of threads.

Value

Invisibly returns `NULL`.

`simulate_hmnl_data` *Simulate hierarchical multinomial logit data*

Description

Generates synthetic panel choice data from the hierarchical (random coefficients + alternative-level random effects) logit DGP: respondents `i = 1..N` face `T` choice situations each, with utilities

$$U_{ijt} = x'_{ijt}\gamma_i + \delta_j + \epsilon_{ijt}, \qquad U_{iot} = \epsilon_{iot},$$

i.i.d. Gumbel shocks (including a shock on the outside option, whose systematic utility is 0), $\beta_i \sim N(\beta, W)$ with $\gamma_{ik} = \beta_{ik}$ or $\exp(\beta_{ik})$ per `rc_dist`, and $\delta_j = z'_j\theta + \xi_j$ with $\xi_j \sim N(0, \sigma_d^2)$. Covariates are Uniform(-1, 1); the alternative-level covariates `z*` are constant within each alternative.

Usage

```
simulate_hmnl_data(
  N = 500,
  T = 10,
  J = 4,
  beta = c(0.8, -0.6),
  W = NULL,
  theta = c(0.5, -0.4),
  sigma_d = 0.5,
  Z = NULL,
  rc_dist = NULL,
  include_outside = TRUE,
  seed = 123,
  vary_choice_set = FALSE
)
```

Arguments

N	Number of respondents.
T	Number of choice situations per respondent.
J	Number of inside alternatives.
beta	Population means of the structural random coefficients (length K_x = length(beta); chain scale for log-normal coordinates).
W	Covariance of the random coefficients (K_x x K_x). Defaults to diag(0.5, K_x).
theta	Mean-function coefficients for $\delta_j = z_j'\theta + \xi_j$; the first entry is the intercept, entries 2..P load on the alternative-level covariates.
sigma_d	Standard deviation of the alternative-level effects ξ_j.
Z	Optional J x (length(theta) - 1) matrix of alternative-level covariates (excluding the intercept). Default NULL draws them Uniform(-1, 1).
rc_dist	Integer vector (length K_x): 0L for normal, 1L for log-normal coordinates. Default NULL is all-normal.
include_outside	Logical; if TRUE (default) every choice set also contains an outside option with systematic utility 0 and its own Gumbel shock. The outside good is *implicit* in the returned data (matching the prepare_hmnl_data() convention): no physical row is emitted, and a choice situation the outside option wins has an all-zeros choice column.
seed	Random seed (NULL skips set.seed()).
vary_choice_set	Logical; if TRUE choice-set size is sampled uniformly from 2:J per task. Default FALSE.

Details

Log-normal coordinates are reported on the *chain* (log) scale in true_params$beta — the scale on which the estimator's hierarchy operates — while entering utility as exp(beta_ik).

Value

A `choicer_sim` object. `true_params` contains `beta`, `W`, `theta`, `sigma_d`, the realized `delta` and `xi` vectors, the full mean-function design `Z` (intercept first), and `rc_dist`.

Examples

```
sim <- simulate_hmnl_data(N = 100, T = 4, J = 4, seed = 123)
print(sim)
sim$true_params$delta
```

`simulate_hmnp_data` *Simulate hierarchical multinomial probit data*

Description

Generates synthetic panel choice data from the hierarchical probit DGP with iid normal utility shocks:

$$U_{ijt} = x'_{ijt}\beta_i + \delta_j + \epsilon_{ijt}, \qquad U_{iot} = \epsilon_{iot}, \qquad \epsilon \sim N(0, \sigma^2),$$

choice by argmax within the task. The outside option is stochastic — it carries its own $N(0, \sigma^2)$ shock on top of systematic utility 0, exactly as in the estimator. $\beta_i \sim N(\beta, W)$ (normal only) and $\delta_j = z'_j\theta + \xi_j, \xi_j \sim N(0, \sigma_d^2)$, as in `simulate_hmnl_data()`.

Usage

```
simulate_hmnp_data(
  N = 500,
  T = 10,
  J = 4,
  beta = c(0.8, -0.6),
  W = NULL,
  theta = c(0.5, -0.4),
  sigma_d = 0.5,
  Z = NULL,
  include_outside = TRUE,
  seed = 123,
  vary_choice_set = FALSE,
  sigma = 1
)
```

Arguments

`N`	Number of respondents.
`T`	Number of choice situations per respondent.
`J`	Number of inside alternatives.

beta	Population means of the structural random coefficients (length K_x = length(beta)); all coordinates are normal.
W	Covariance of the random coefficients (K_x x K_x). Defaults to diag(0.5, K_x).
theta	Mean-function coefficients for $\delta_j = z_j'\theta + \xi_j$; the first entry is the intercept, entries 2..P load on the alternative-level covariates.
sigma_d	Standard deviation of the alternative-level effects ξ_j.
Z	Optional J x (length(theta) - 1) matrix of alternative-level covariates (excluding the intercept). Default NULL draws them Uniform(-1, 1).
include_outside	Logical; if TRUE (default) every choice set also contains an outside option with systematic utility 0 and its own normal shock. The outside good is *implicit* in the returned data (matching the prepare_hmnp_data() convention): no physical row is emitted, and a choice situation the outside option wins has an all-zeros choice column.
seed	Random seed (NULL skips set.seed()).
vary_choice_set	Logical; if TRUE choice-set size is sampled uniformly from 2:J per task. Default FALSE.
sigma	Standard deviation of the iid utility shocks (DGP scale).

Details

The iid-probit likelihood identifies parameters only up to the common scale σ, so true_params is reported on the *identified* scale: beta $= \beta/\sigma$, W $= W/\sigma^2$, theta $= \theta/\sigma$, sigma_d $= \sigma_d/\sigma$, delta $= \delta/\sigma$, xi $= \xi/\sigma$. With the default sigma = 1 the DGP scale and the identified scale coincide.

Value

A choicer_sim object. true_params contains beta, W, theta, sigma_d, the realized delta and xi, and the full mean-function design Z — all on the identified scale (see Details).

Examples

```
sim <- simulate_hmnp_data(N = 100, T = 4, J = 4, seed = 123)
print(sim)
sim$true_params$delta
```

---

simulate_mnl_data *Simulate multinomial logit data*

Description

Generates synthetic choice data with i.i.d. Gumbel errors, optionally with varying choice-set sizes and an outside option (alt = 0). Choices are determined by argmax of utility; covariates are drawn as Uniform(-1, 1).

Usage

```
simulate_mnl_data(
  N = 5000,
  J = 5,
  beta = c(0.8, -0.6),
  delta = NULL,
  seed = 123,
  outside_option = TRUE,
  vary_choice_set = TRUE
)
```

Arguments

N	Number of choice situations.
J	Number of inside alternatives.
beta	Fixed coefficients for x1..x{K_x} (length K_x = length(beta)).
delta	Alternative-specific constants for inside alternatives (length J). Defaults to an alternating pattern of c(0.5, -0.5).
seed	Random seed. Pass NULL to skip set.seed() (useful inside monte_carlo() where the caller manages RNG).
outside_option	Logical; if TRUE (default) an outside option with alt = 0 and zero covariates is added to every choice set.
vary_choice_set	Logical; if TRUE (default) choice set size is sampled uniformly from 2:J; if FALSE every individual faces all J inside alternatives.

Value

A choicer_sim object.

Examples

```
sim <- simulate_mnl_data(N = 1000, J = 5, seed = 123)
print(sim)
```

---

simulate_mnp_data *Simulate multinomial probit data*

Description

Generates synthetic choice data from the MNP data-generating process estimated by `run_mnprobit()`: latent utility differences against the base alternative (alternative 1),

$$w_i = X_i\beta + \delta + \varepsilon_i, \qquad \varepsilon_i \sim N_{J-1}(0, \Sigma),$$

with alternative $j > 1$ chosen iff $w_{ij} > \max(0, \max_{k \neq j} w_{ik})$ and the base chosen iff all $w_{ij} < 0$. Covariates are Uniform(-1, 1). Choice sets are balanced (every individual faces all `J` alternatives), as the MNP estimator requires; there is no outside-option flag — model an outside good as a zero-covariate base alternative instead.

Usage

```
simulate_mnp_data(
  N = 5000,
  J = 3,
  beta = c(0.8, -0.6),
  delta = NULL,
  Sigma = matrix(c(1, 0.5, 0.5, 1.5), nrow = 2),
  seed = 123
)
```

Arguments

`N`	Number of choice situations.
`J`	Number of alternatives (alternative 1 is the base).
`beta`	Fixed coefficients for `x1..x{K_x}` (length `K_x = length(beta)`).
`delta`	ASCs of the differenced utilities, one per non-base alternative (length `J - 1`). Defaults to an alternating pattern of `c(0.5, -0.5)`.
`Sigma`	Covariance matrix of the differenced errors (`(J-1) x (J-1)`).
`seed`	Random seed (`NULL` skips `set.seed()`).

Details

The MNP likelihood only identifies parameters up to scale, so `true_params` is reported on the *identified* scale (normalized by σ_{11}): `beta` $= \beta/\sqrt{\sigma_{11}}$, `delta` $= \delta/\sqrt{\sigma_{11}}$, and `Sigma` $= \Sigma/\sigma_{11}$ — the scale on which `run_mnprobit()` reports its posterior. With the default `Sigma` ($\sigma_{11} = 1$) the DGP scale and the identified scale coincide.

Value

A `choicer_sim` object. `true_params` contains `beta`, `delta`, and `Sigma` on the identified scale (see Details).

Examples

```
sim <- simulate_mnp_data(N = 1000, J = 3, seed = 123)
print(sim)
```

`simulate_mxl_data` *Simulate mixed logit data*

Description

Generates synthetic choice data with random coefficients drawn from a multivariate normal (optionally log-normal per dimension) and an additional mean shifter `mu`. Random coefficients are parameterized via the lower Cholesky factor of `Sigma`. Covariates are Uniform(-1, 1) by default; columns named in `price_cols` are drawn as `-Uniform(0.1, 3)` to mimic strictly-negative price variables.

Usage

```
simulate_mxl_data(
  N = 5000,
  J = 4,
  beta = c(0.8, -0.6),
  delta = NULL,
  mu = NULL,
  Sigma = matrix(c(1, 0.5, 0.5, 1.5), nrow = 2),
  rc_dist = NULL,
  rc_correlation = NULL,
  price_cols = NULL,
  seed = 123,
  outside_option = TRUE,
  vary_choice_set = TRUE
)
```

Arguments

`N`	Number of choice situations.
`J`	Number of inside alternatives.
`beta`	Fixed coefficients for `x1..x{K_x}` (length `K_x = length(beta)`).
`delta`	ASCs for inside alternatives (length `J`). Defaults to an alternating pattern of `c(0.5, -0.5)`.
`mu`	Mean shifter for random coefficients (length `K_w = ncol(Sigma)`). Defaults to a zero vector.
`Sigma`	Covariance matrix of random coefficients (square, `K_w x K_w`).
`rc_dist`	Integer vector (length `K_w`): `0L` for normal, `1L` for log-normal. Default `NULL` is treated as all-normal.
`rc_correlation`	Logical; if `NULL` (default) it is auto-detected from the off-diagonal entries of `Sigma`.
`price_cols`	Character vector of `w*` column names to draw as `-Uniform(0.1, 3)` instead of `Uniform(-1, 1)`. Default `NULL`.

seed	Random seed (NULL skips set.seed()).
outside_option	Logical; include outside option with alt = 0.
vary_choice_set	Logical; if TRUE (default) choice set size is sampled uniformly from 2:J.

Details

Random coefficients are constructed to match the estimator's parameterization in src/mxlogit.cpp. For every dimension the raw draw is L %*% eta where eta ~ N(0, I). A normal random coefficient (rc_dist = 0) is then gamma_k = mu_k + (L %*% eta)_k. A log-normal random coefficient (rc_dist = 1) follows the shifted log-normal beta_k = exp(mu_k) + exp((L %*% eta)_k) – not the textbook exp(mu_k + sigma_k * eta) – so mu_k in true_params$mu is on the same scale the estimator recovers and recovery_table() can compare like-for-like.

Value

A choicer_sim object. true_params includes beta, delta, Sigma, L_params (packed Cholesky parameters), mu, rc_dist, rc_correlation.

Examples

```
sim <- simulate_mxl_data(N = 1000, J = 4, seed = 123)
print(sim)
```

---

simulate_nl_data *Simulate nested logit data*

Description

Generates synthetic choice data with nested logit probabilities computed analytically (log-sum-exp over inclusive values), then samples choices from the implied multinomial. The outside option (j = 0) sits in a singleton nest with lambda = 1.

Usage

```
simulate_nl_data(
  N = 10000,
  beta = c(1.5, -0.8),
  delta = c(`1` = 0.5, `2` = 0.3, `3` = -0.2, `4` = -0.5, `5` = 0.4),
  nests = list(c(1, 2), c(3, 4, 5)),
  lambdas = c(0.8, 0.2),
  seed = 123
)
```

Arguments

N	Number of choice situations.
beta	Fixed coefficients for covariates X, W (length 2 by default).
delta	Named numeric vector of ASCs for inside alternatives.
nests	List of integer vectors defining nest membership for inside alternatives.
lambdas	Numeric vector of dissimilarity parameters, one per nest.
seed	Random seed (NULL skips set.seed()).

Value

A choicer_sim object. true_params includes beta, delta, lambdas; settings includes the nest_structure. The returned data retains a nest column (integer, with 0L for the outside option) for convenient use with run_nestlogit().

Note

Unlike simulate_mnl_data() and simulate_mxl_data(), this function does not expose outside_option or vary_choice_set flags. The outside option (j = 0) is always present as a singleton nest with lambda = 1, and every individual faces the full set of inside alternatives. Add these flags if downstream use cases need them.

Examples

```
sim <- simulate_nl_data(N = 2000, seed = 123)
print(sim)
```

---

summary.choicer_hb *Summarize a hierarchical Bayes fit*

Description

Posterior summaries (mean, SD, equal-tailed credible interval) for the population coefficients b, the mean-function coefficients θ, the alternative-effect variance σ_d^2 (and, for the HMNP, the raw shock variance trace), plus the δ_j / ξ_j quality ladder, acceptance diagnostics, and a consolidated convergence-diagnostic table (rank-normalized R-hat, ESS bulk/tail, MCSE) built from all retained chains (see rhat(), ess(), mcse()).

Usage

```
## S3 method for class 'choicer_hb'
summary(object, prob = 0.95, ...)
```

Arguments

object	A choicer_hmnl or choicer_hmnp object.
prob	Probability mass of the equal-tailed credible interval (default 0.95).
...	Additional arguments (ignored).

Value

A summary.choicer_hb object.

Examples

```
sim <- simulate_hmnl_data(N = 50, T = 2, J = 3, seed = 42)
fit <- suppressWarnings(run_hmnlogit(sim$data, "task", "alt", "choice", c("x1", "x2"),
                    person_col = "pid",
                    mcmc = list(R = 300, burn = 100)))
summary(fit)
```

---

summary.choicer_mnl *Summary for multinomial logit model*

Description

Computes and returns a coefficient summary table with standard errors, z-values, p-values, and significance codes. Triggers lazy Hessian computation if standard errors have not been computed yet.

Usage

```
## S3 method for class 'choicer_mnl'
summary(object, gof = TRUE, ...)
```

Arguments

object	A choicer_mnl object.
gof	Logical; compute goodness-of-fit measures (McFadden R-squared, hit rate) for the summary footer. Involves an in-sample prediction pass (for mixed logit, a full simulation over draws); set to FALSE to skip.
...	Additional arguments (ignored).

Value

A summary.choicer_mnl object (list with coefficients table and metadata, including a gof element with goodness-of-fit measures from gof; its fields are NA when the model was fitted with keep_data = FALSE).

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
summary(fit)
```

`summary.choicer_mnp` *Summary for Bayesian multinomial probit model*

Description

Posterior summaries (mean, SD, equal-tailed credible interval) of the identified coefficient and covariance draws.

Usage

```
## S3 method for class 'choicer_mnp'
summary(object, prob = 0.95, ...)
```

Arguments

`object`	A choicer_mnp object.
`prob`	Probability mass of the equal-tailed credible interval (default 0.95).
`...`	Additional arguments (ignored).

Value

A summary.choicer_mnp object (list with coefficient and Sigma posterior tables plus metadata).

Examples

```
library(data.table)
set.seed(42)
N <- 100; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnprobit(dt, "id", "alt", "choice", c("x1", "x2"),
                    mcmc = list(R = 300, burn = 100))
summary(fit)
```

`summary.choicer_mxl` *Summary for mixed logit model*

Description

Computes coefficient summary with delta-method transformation for variance parameters (Cholesky to covariance scale) and log-normal mean parameters. Triggers lazy Hessian computation if standard errors have not been computed yet.

Usage

```
## S3 method for class 'choicer_mxl'
summary(object, gof = TRUE, ...)
```

Arguments

`object`	A choicer_mxl object.
`gof`	Logical; compute goodness-of-fit measures (McFadden R-squared, hit rate) for the summary footer. Involves an in-sample prediction pass (for mixed logit, a full simulation over draws); set to FALSE to skip.
`...`	Additional arguments (ignored).

Value

A summary.choicer_mxl object (includes a `gof` element with goodness-of-fit measures from `gof`; its fields are NA when the model was fitted with `keep_data = FALSE`).

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), w1 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mxlogit(
  data = dt, id_col = "id", alt_col = "alt", choice_col = "choice",
  covariate_cols = "x1", random_var_cols = "w1", S = 50L
)
summary(fit)
```

`summary.choicer_nl` *Summary for nested logit model*

Description

Triggers lazy Hessian computation if standard errors have not been computed yet.

Usage

```
## S3 method for class 'choicer_nl'
summary(object, gof = TRUE, ...)
```

Arguments

`object`	A choicer_nl object.
`gof`	Logical; compute goodness-of-fit measures (McFadden R-squared, hit rate) for the summary footer. Involves an in-sample prediction pass (for mixed logit, a full simulation over draws); set to FALSE to skip.
`...`	Additional arguments (ignored).

Value

A summary.choicer_nl object (includes a `gof` element with goodness-of-fit measures from `gof`; its fields are NA when the model was fitted with `keep_data = FALSE`).

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 4
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, nest := ifelse(alt <= 2, "A", "B")]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_nestlogit(
  data = dt, id_col = "id", alt_col = "alt", choice_col = "choice",
  covariate_cols = c("x1", "x2"), nest_col = "nest"
)
summary(fit)
```

thread_info *Query choicer OpenMP thread settings*

Description

Query choicer OpenMP thread settings

Usage

```
thread_info()
```

Value

A list with OpenMP availability, active/max thread settings, CPU thread capacity reported by OpenMP, thread limits, and relevant environment variables.

traceplot *Traceplot for a hierarchical Bayes fit*

Description

Generic dispatching on the fit's class. See `traceplot.choicer_hb()` for the `choicer_hmnl` / `choicer_hmnp` method.

Usage

```
traceplot(object, ...)
```

Arguments

`object`	A fitted model object.
`...`	Additional arguments passed to methods.

Value

The object, invisibly.

`traceplot.choicer_hb` *Traceplot method for hierarchical Bayes fits*

Description

Overlaid per-chain traceplots (base `graphics`, no new dependency) for a `choicer_hmnl` / `choicer_hmnp` fit's population coefficients (`b`), delta mean-function coefficients (`theta`), alternative-effect variance (`sigma_d2`), and, opt-in, a representative subset of the (potentially ~200-column) alternative effects (`delta`).

Usage

```
## S3 method for class 'choicer_hb'
traceplot(object, block = c("b", "theta", "sigma_d2"), which = NULL, ...)
```

Arguments

`object`	A `choicer_hmnl` or `choicer_hmnp` fit.
`block`	Character vector, any non-empty subset of `c("b", "theta", "sigma_d2", "delta")` (HMNP fits additionally accept `"sigma2"`, the raw non-identified shock-variance chain). Default plots `b`, `theta`, `sigma_d2` (not `delta`, which can have ~200 columns).
`which`	Only consulted when `"delta"` is in `block`. `NULL` (default) auto-selects a representative subset (the 3 highest rank-R-hat plus 3 lowest ESS_bulk alternatives, deduplicated, capped at 6). If supplied, a character vector of alternative labels or an integer vector of column indices.
`...`	Additional arguments (ignored).

Value

`object`, invisibly.

Examples

```
sim <- simulate_hmnl_data(N = 60, T = 2, J = 4, seed = 1)
fit <- suppressWarnings(run_hmnlogit(sim$data, "task", "alt", "choice", c("x1", "x2"),
                    person_col = "pid",
                    mcmc = list(R = 300, burn = 100), chains = 2))
traceplot(fit)
```

`vcov.choicer_fit` *Extract variance-covariance matrix from a choicer_fit object*

Description

With no arguments, returns the variance-covariance matrix implied by the fit's own `se_method` (triggering lazy computation if needed). Passing `type` recomputes a different variance estimator post hoc from the stored data — no refit needed (requires `keep_data = TRUE`):

`"hessian"` Inverse of the analytical negated Hessian.

`"bhhh"` Inverse of the BHHH/OPG information $\sum_i w_i s_i s_i'$.

`"robust"` Huber-White sandwich $A^{-1}(\sum_i w_i^2 s_i s_i')A^{-1}$ — also the valid WESML variance under choice-based weighting.

`"cluster"` Cluster-robust sandwich $A^{-1}(\sum_g g_g g_g')A^{-1}$ with $g_g = \sum_{i \in g} w_i s_i$ the within-cluster sum of weighted scores. Requires `cluster` (or a fit made with `cluster_col`). No small-sample correction is applied.

Here i indexes *choice situations*. For repeated choices by the same decision maker (panel data), cluster on the decision maker.

Usage

```
## S3 method for class 'choicer_fit'
vcov(object, type = NULL, cluster = NULL, ...)
```

Arguments

`object`	A choicer_fit object.
`type`	`NULL` (default; return the as-fitted vcov) or one of `"hessian"`, `"bhhh"`, `"robust"`, `"cluster"`.
`cluster`	Cluster labels for `type = "cluster"`, one per choice situation. Alignment to the prepared (id-sorted) choice situations is handled as follows: • **Named** (recommended): names are matched against the choice-situation ids, so the vector is safe in any order. Build it by naming your per-situation labels with the id values. • **Unnamed**: taken to be in the prepared, id-sorted order; a warning flags that assumption. A vector of per-alternative (row-level) length is rejected. Defaults to the labels stored at fit time via `cluster_col` (already aligned). Supplying `cluster` without `type` implies `type = "cluster"`. The safest route is to pass `cluster_col=` at fit time, which sidesteps post-hoc alignment entirely.
`...`	Additional arguments (ignored).

Details

Note (mixed logit): clustering repairs the *inference*, not the *estimand*. `run_mxlogit()` treats each choice situation as an independent draw from the mixing distribution (a cross-sectional MSL likelihood, not the panel product form), so on panel data the point estimates target that cross-sectional model; `type = "cluster"` makes their standard errors robust to within-person dependence but does not turn the fit into a panel mixed logit. For panel random coefficients use `run_hmnlogit(person_col)`.

Value

Named variance-covariance matrix, or NULL if unavailable.

Examples

```
library(data.table)
set.seed(42)
N <- 50; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, person := rep(1:10, each = 5)[id]]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnlogit(dt, "id", "alt", "choice", c("x1", "x2"))
vcov(fit)                          # as fitted (hessian)
vcov(fit, type = "robust")         # Huber-White, post hoc
# named by situation id -> safe regardless of order
cl <- dt[, person[1L], by = id]
vcov(fit, type = "cluster", cluster = setNames(cl$V1, cl$id))
```

`vcov.choicer_hb` *Posterior covariance of the population coefficients*

Description

Posterior covariance of the population coefficients

Usage

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

Arguments

`object`	A `choicer_hmnl` or `choicer_hmnp` object.
`...`	Additional arguments (ignored).

Value

K x K posterior covariance matrix of the b draws.

`vcov.choicer_mnp` *Extract variance-covariance matrix from a choicer_mnp object*

Description

Returns the posterior covariance matrix of the identified coefficient draws (computed eagerly at fit time; no Hessian is involved).

Usage

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

Arguments

`object`	A choicer_mnp object.
`...`	Additional arguments (ignored).

Value

Named posterior covariance matrix.

Examples

```
library(data.table)
set.seed(42)
N <- 100; J <- 3
dt <- data.table(id = rep(1:N, each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), x2 = rnorm(.N))]
dt[, choice := 0L]
dt[, choice := sample(c(1L, rep(0L, J - 1))), by = id]
fit <- run_mnprobit(dt, "id", "alt", "choice", c("x1", "x2"),
                    mcmc = list(R = 300, burn = 100))
vcov(fit)
```

`wesml_vcov` *Robust (sandwich) variance for a weighted / choice-based logit fit*

Description

Recomputes the robust Huber-White sandwich variance $V = A^{-1}BA^{-1}$ for a fitted multinomial (MNL), mixed (MXL) or nested (NL) logit, where the bread $A = \sum_i w_i(-H_i)$ is the weighted negated Hessian and the meat $B = \sum_i w_i^2 s_i s_i'$ is the weight-squared outer product of the per-individual scores. This is the appropriate variance under choice-based (endogenous stratified) / WESML weighting, where the inverse-Hessian and the ordinary BHHH variance are invalid. It can be called on any fitted model (e.g. one estimated with `se_method = "hessian"`) to obtain robust standard errors post hoc, without refitting.

Usage

```
wesml_vcov(object, ...)

## S3 method for class 'choicer_mxl'
wesml_vcov(object, type = c("vcov", "se"), ...)

## S3 method for class 'choicer_mnl'
wesml_vcov(object, type = c("vcov", "se"), ...)

## S3 method for class 'choicer_nl'
wesml_vcov(object, type = c("vcov", "se"), ...)
```

Arguments

`object`	A fitted `choicer_mnl`, `choicer_mxl` or `choicer_nl` object (requires `keep_data = TRUE`).
`...`	Unused.
`type`	Either `"vcov"` (default) to return the variance-covariance matrix or `"se"` to return the standard-error vector.

Details

If the stored weights are uniform (all equal), a warning is emitted: the returned variance is then the ordinary robust (Huber-White) variance, not a WESML-weighted variance. Refit with WESML weights for a choice-based-sampling correction.

Value

A variance-covariance matrix (`type = "vcov"`) or a named numeric vector of standard errors (`type = "se"`), in the raw parameter space.

See Also

`wesml_weights`, `sample_by_choice`, `run_mxlogit`

Examples

```
library(data.table)
set.seed(1)
N <- 200L; J <- 3L
dt <- data.table(id = rep(seq_len(N), each = J), alt = rep(1:J, N))
dt[, `:=`(x1 = rnorm(.N), w1 = rnorm(.N))]
dt[, choice := as.integer(seq_len(.N) == sample.int(.N, 1L)), by = id]
fit <- run_mxlogit(dt, "id", "alt", "choice", "x1", "w1", S = 50L)
wesml_vcov(fit, "se")
```

`wesml_weights` *WESML weights for choice-based (endogenous stratified) samples*

Description

Computes Manski-Lerman (1977) Weighted Exogenous Sample Maximum Likelihood (WESML) weights for a choice-based sample. The weight for a choice situation whose chosen alternative is j is $w = Q(j)/H(j)$, where $Q(j)$ is the population share of alternative j and $H(j)$ its sample share among choosers. Using these weights in `run_mxlogit` restores consistency under choice-based sampling; pair them with `se_method = "sandwich"` for valid (robust) standard errors (the plain inverse-Hessian is invalid under weighting).

Usage

```
wesml_weights(
  data,
  id_col,
  alt_col,
  choice_col,
  Q,
  H = NULL,
  normalize = TRUE,
  attach = FALSE,
  weight_name = ".wesml_weight",
  outside_opt_label = NULL,
  include_outside_option = FALSE
)
```

Arguments

`data`	A long-format choice data set (data.frame or data.table), one row per alternative per choice situation.
`id_col, alt_col, choice_col`	Column names identifying the choice situation, the alternative, and the 0/1 chosen indicator.

Q	Named numeric vector of population shares, one entry per chosen stratum (names matched to as.character(alt)), each strictly positive. Renormalized to sum 1 if needed.
H	Optional named numeric vector of sample shares. If NULL (default) it is computed from data as the fraction of choice situations choosing each alternative.
normalize	If TRUE (default) the returned weights are scaled to mean 1. This does not affect the point estimates or the sandwich variance.
attach	If TRUE, return data with a row-level weight column appended (the per-situation weight repeated across all rows of a situation), ready to pass to run_mxlogit(weights_col = ...). If FALSE (default) return an id-keyed table of weights.
weight_name	Name of the weight column (default ".wesml_weight").
outside_opt_label, include_outside_option	Set include_outside_option = TRUE and supply outside_opt_label when the outside good is implicit (choice situations with no 1 in choice_col are treated as having chosen the outside good).

Details

Strata are defined by the chosen alternative and keyed by as.character(alt) so numeric and character alternative codes match supplied share names unambiguously.

Value

Either an id-keyed data.table with columns id_col and weight_name (default), or, when attach = TRUE, a copy of data with the weight column appended. The result carries "Q", "H", and "choice_sampling" attributes recording provenance.

References

Manski, C. F. and Lerman, S. R. (1977). The Estimation of Choice Probabilities from Choice Based Samples. *Econometrica* 45(8), 1977-1988. Train, K. E. (2009). *Discrete Choice Methods with Simulation*, Section 3.7. Cambridge University Press.

See Also

sample_by_choice, run_mxlogit, wesml_vcov

Examples

```
library(data.table)
set.seed(1)
N <- 300L; J <- 3L
pop <- data.table(id = rep(seq_len(N), each = J), alt = rep(1:J, N))
pop[, x1 := rnorm(.N)]
pop[, w1 := rnorm(.N)]
pop[, choice := as.integer(seq_len(.N) == sample.int(.N, 1L)), by = id]

# Population shares of the chosen alternative
Q <- prop.table(table(pop[choice == 1, alt]))
```

```
wt <- wesml_weights(pop, "id", "alt", "choice", Q = Q)
head(wt)
```

`wtp.choicer_hb` *Compute willingness to pay*

Description

Computes willingness-to-pay (WTP) ratios with delta-method standard errors from a fitted choice model. For an attribute coefficient θ_k and a price coefficient θ_p, the WTP is

$$WTP_k = -\theta_k/\theta_p,$$

the marginal rate of substitution between the attribute and price. Standard errors use the delta method with analytic gradients $\partial g/\partial\theta_k = -1/\theta_p$ and $\partial g/\partial\theta_p = \theta_k/\theta_p^2$, applied to the corresponding 2x2 block of `vcov(object)`.

Usage

```
## S3 method for class 'choicer_hb'
wtp(object, price_var, attr_vars = NULL, level = 0.95, ...)

wtp(object, price_var, attr_vars = NULL, level = 0.95, ...)

## S3 method for class 'choicer_fit'
wtp(object, price_var, attr_vars = NULL, level = 0.95, ...)

## S3 method for class 'choicer_mxl'
wtp(object, price_var, attr_vars = NULL, level = 0.95, ...)
```

Arguments

`object`	A fitted model object (`choicer_mnl`, `choicer_mxl`, or `choicer_nl`).
`price_var`	Name of the price variable. Must be a fixed-coefficient variable (a column of the design matrix `X`).
`attr_vars`	Character vector of attributes to report. Defaults to all fixed-coefficient variables other than `price_var` (plus, for mixed logit with `rc_mean = TRUE`, all random coefficients). ASC names (e.g. `"ASC_2"`) may also be supplied; the WTP of an ASC is $-ASC_j/\theta_p$.
`level`	Confidence level for the normal-approximation interval $Estimate \pm z_{1-(1-level)/2} \times SE$. Default 0.95.
`...`	Additional arguments passed to methods.

Details

For mixed logit models, random coefficients are included via their estimated location parameters. The package's log-normal random coefficient is the *shifted* log-normal $\beta_k = \exp(\mu_k) + \exp((L\eta)_k)$ (see `run_mxlogit()`), so:

- Normal random coefficient k (`rc_mean = TRUE`): mean WTP $-\mu_k/\theta_p$, labeled `Mu_x`.
- Log-normal random coefficient k (`rc_mean = TRUE`): **median** WTP $-(\exp(\mu_k)+1)/\theta_p$, since the median of $\exp((L\eta)_k)$ is 1. (The mean, $\exp(\mu_k) + \exp(\sigma_k^2/2)$, is highly sensitive to the estimated variance; the median is the more robust summary.) These rows are labeled by the attribute name and flagged as medians when printed.
- Log-normal random coefficient with `rc_mean = FALSE`: $\beta_k = \exp((L\eta)_k)$ has median 1, so the median WTP is $-1/\theta_p$ with uncertainty driven solely by θ_p.

Normal random coefficients with `rc_mean = FALSE` have mean 0 by construction and are excluded from the table.

The price variable must have a *fixed* coefficient. A random price coefficient is rejected: the ratio of two random coefficients generally has no finite moments (the denominator has positive density at 0), so mean or median WTP computed from location parameters would be meaningless. In `choicer`, use a fixed price coefficient. WTP-space estimation is not currently implemented; it is an alternative specification available in other software rather than an option supplied by this function.

Value

A data.frame of class `choicer_wtp` with one row per attribute and columns `Estimate`, `Std_Error`, `z_value`, `CI_lower`, `CI_upper`. Attributes `price_var` and `level` record the inputs; `median_rows` lists rows that are median (rather than mean) WTP. Standard errors are NA when the variance-covariance matrix is unavailable.

Methods (by class)

- `wtp(choicer_hb)`: Posterior willingness-to-pay for hierarchical Bayes fits: the per-draw ratio of population-mean utility coefficients, $-\bar{\gamma}_{attr}/\bar{\gamma}_{price}$ (for log-normal coordinates $\bar{\gamma} = \exp(b + W_{kk}/2)$). Ratio posteriors are heavy-tailed, so the point estimate is the posterior **median** with equal-tailed quantile intervals — never a posterior mean or a delta-method SE. A warning is raised when the price coefficient's sign is not resolved by the posterior. If the price variable was flagged as endogenous-without-a-control-function at prep time, WTP inherits that caveat (see `cf_residual_col` in `prepare_hmnl_data()`).

Examples

```
library(data.table)
sim <- simulate_mnl_data(N = 1000, J = 4, beta = c(0.8, -0.6), seed = 123,
                         outside_option = FALSE, vary_choice_set = FALSE)
fit <- run_mnlogit(sim$data, "id", "alt", "choice", c("x1", "x2"))
# treat x2 as the price variable
wtp(fit, price_var = "x2")
wtp(fit, price_var = "x2", attr_vars = c("x1", "ASC_2"), level = 0.90)
```

Index

∗ **datasets**
    mode_choice, 29

blp, 4
blp.choicer_mnl, 4
blp.choicer_mxl, 5
blp.choicer_nl, 7
blp_contraction, 8

coef.choicer_fit, 9
coef.choicer_hb, 10
coef.choicer_mnp, 10
consumer_surplus, *26*
consumer_surplus *(*consumer_surplus.choicer_hmnl*)*, 11
consumer_surplus.choicer_hmnl, 11

diversion_ratios *(*diversion_ratios.choicer_hb*)*, 14
diversion_ratios.choicer_hb, 14
diversion_ratios.choicer_mnl, 15
diversion_ratios.choicer_mxl, 16
diversion_ratios.choicer_nl, 17

elasticities *(*elasticities.choicer_hb*)*, 18
elasticities.choicer_hb, 18
elasticities.choicer_hb(), *15*
elasticities.choicer_mnl, 19
elasticities.choicer_mxl, 20
elasticities.choicer_nl, 21
ess, 22, *29*
ess(), *67*, *69, 70*, *91*

fft, *22*

get_halton_normals, 22, *78*
gof, 23, *92*, *94, 95*

logLik.choicer_fit, 24
logsum, *11*, *13, 14*
logsum *(*logsum.choicer_hmnl*)*, 25
logsum.choicer_hmnl, 25
logsum.choicer_hmnp(), *14*

mc_asymptotics, 27
mcse, 28
mcse(), *67*, *69, 70*, *91*
mode_choice, 29
monte_carlo, 31
monte_carlo(), *27*
mxl_blp_contraction, 32
mxl_logsum, *25*

new_choicer_sim, 34
nl_blp_contraction, 34
nobs.choicer_fit, 36
nobs.choicer_hb, 37
nobs.choicer_mnp, 37

ppc_shares, 38
predict.choicer_hb, 39
predict.choicer_hb(), *38*
predict.choicer_mnl, 40
predict.choicer_mxl, 41
predict.choicer_nl, 43
prepare_hmnl_data, 44
prepare_hmnl_data(), *47, 48*, *66, 67*, *84*, *105*
prepare_hmnp_data, 47
prepare_hmnp_data(), *46*, *69, 70*, *86*
prepare_mnl_data, 49, *54, 55*, *71, 72*
prepare_mnl_data(), *45*
prepare_mnp_data, 50, *73, 74*
prepare_mnp_data(), *48*
prepare_mxl_data, 52, *76*, *78*
prepare_nl_data, 54, *81*
print.choicer_cs, 55
print.choicer_fit, 56
print.choicer_gof, 56

print.choicer_hb, 57
print.choicer_mnp, 58
print.choicer_wtp, 58
print.summary.choicer_hb, 59
print.summary.choicer_mnl, 59
print.summary.choicer_mnp, 60
print.summary.choicer_mxl, 61
print.summary.choicer_nl, 62

recovery_table, 63
recovery_table(), *34*, *67*
rhat, 64
rhat(), *67*, *91*
run_hmnlogit, 65, *77*, *99*
run_hmnlogit(), *69, 70*
run_hmnprobit, 68
run_mnlogit, 70, *77*, *80*
run_mnprobit, 73
run_mnprobit(), *88*
run_mxlogit, 75, *82*, *101–103*
run_nestlogit, 79
run_nestlogit(), *91*

sample_by_choice, *78*, 81, *101*, *103*
set_num_threads, 83
simulate_hmnl_data, 83
simulate_hmnl_data(), *67*, *85*
simulate_hmnp_data, 85
simulate_hmnp_data(), *64*, *70*
simulate_mnl_data, 86
simulate_mnl_data(), *34*, *91*
simulate_mnp_data, 87
simulate_mnp_data(), *64*
simulate_mxl_data, 89
simulate_mxl_data(), *34*, *91*
simulate_nl_data, 90
simulate_nl_data(), *34*
summary.choicer_hb, 91
summary.choicer_mnl, 92
summary.choicer_mnp, 93
summary.choicer_mxl, 94
summary.choicer_nl, 95

thread_info, 96
traceplot, 96
traceplot(), *67*, *69, 70*
traceplot.choicer_hb, 97
traceplot.choicer_hb(), *96*

vcov.choicer_fit, 98
vcov.choicer_hb, 99
vcov.choicer_mnp, 100

wesml_vcov, 101, *103*
wesml_weights, *30*, *78*, *81, 82*, *101*, 102
wtp, *13, 14*
wtp *(*wtp.choicer_hb*)*, 104
wtp.choicer_hb, 104
wtp.choicer_hb(), *14*