Package 'combreg'

July 29, 2026

Type Package

Title Bayesian Regression for Combinatorial Response Data

Version 0.2.0

Description Bayesian regression whose response data are integer-valued vectors subject to combinatorial constraints in the form of Ay<=b. Implements the Metropolis-Hastings-within-Gibbs sampler of Zheng et al. (2026+) <doi:10.48550/arXiv.2504.11630>, an unconstrained probit baseline for comparison, benchmarking helpers, MCMC and regression diagnostics (effective sample sizes, split-Rhat, structured reports), plotting methods (trace, autocorrelation, violin, effective-sample-size, sampling-efficiency, residual heat map), and utilities for constraint validation (total unimodularity, feasibility) and data simulation.

Depends R (>= 4.0)

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Imports Rcpp, coda, grDevices, graphics, stats, truncnorm, utils

LinkingTo Rcpp, RcppArmadillo

Suggests lpSolve, posterior, testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/YuZh98/combreg

BugReports https://github.com/YuZh98/combreg/issues

NeedsCompilation yes

Author Hugh Zheng [aut, cre, cph]

Maintainer Hugh Zheng <hugh.stats@gmail.com>

Repository CRAN

Date/Publication 2026-07-29 18:40:07 UTC

Contents

`as.mcmc.crr_fit` *Convert to coda mcmc.list*

Description

Post-warmup, thinned draws per chain.

Usage

```
## S3 method for class 'crr_fit'
as.mcmc(x, ...)
```

Arguments

`x`	A `crr_fit` object.
`...`	Unused.

Value

A coda::mcmc.list.

`as_draws.crr_fit` *Convert to posterior draws_array*

Description

Convert to posterior draws_array

Usage

```
## S3 method for class 'crr_fit'
as_draws(x, ...)
```

Arguments

`x`	A `crr_fit` object.
`...`	Unused.

Value

A `posterior::draws_array` (requires the 'posterior' package).

`coef.crr_fit` *Posterior mean coefficients*

Description

Posterior mean coefficients

Usage

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

Arguments

`object`	A `crr_fit` object.
`...`	Unused.

Value

Posterior mean coefficient matrix (`p` x `d`).

Examples

```
if (requireNamespace("lpSolve", quietly = TRUE)) {
  con <- crr_constraints(rbind(c(1, 1, 0), c(0, 1, 1)), b = c(1, 1))
  sim <- simulate_crr(n = 50, p = 2, constraints = con, seed = 1)
  fit <- crr(sim$Y, sim$X, con, n_iter = 200, warmup = 100, seed = 1)
  coef(fit)   # posterior mean of beta, a p x d matrix (row k = covariate k,
              # column j = response coordinate j)
}
```

`coef_precompute` *Precompute quantities for the conjugate coefficient update*

Description

Precompute quantities for the conjugate coefficient update

Usage

```
coef_precompute(X, prior = crr_prior())
```

Arguments

`X`	Covariate matrix (`n` x `p`).
`prior`	A crr_prior object.

Value

List with posterior covariance `V` and its lower Cholesky factor `L`, for use in `update_coef()`.

`crr` *Bayesian combinatorial response regression*

Description

Fits the latent-utility regression model

$$\zeta_i = \beta^\top x_i + \varepsilon_i, \quad \varepsilon_i \sim N(0, I_d), \quad y_i = \arg\max_{y \in \mathcal{Y}} \zeta_i^\top y,$$

where $\mathcal{Y} = \{y \in \{0,1\}^d : Ay \le b\}$ with totally unimodular `A`, by MH-within-Gibbs sampling on the dual-certificate augmented posterior (`method = "mhwg"`). `method = "unconstrained"` fits independent Albert-Chib probit regressions per coordinate, ignoring the constraints — useful as a baseline demonstrating constraint-ignoring bias.

Usage

```
crr(
  Y,
  X,
  constraints,
  data = NULL,
  method = c("mhwg", "unconstrained"),
  kernel = c("exponential", "half_gaussian"),
  prior = crr_prior(),
  n_iter = 5000,
  warmup = floor(n_iter/2),
  thin = 1,
  chains = 1,
  seed = NULL,
  control = crr_control(),
  verbose = FALSE
)
```

Arguments

Y	Response matrix (n x d), rows in $\mathcal{Y}$.
X	Covariate matrix (n x p), or a one-sided formula (e.g. ~ x1 + x2) evaluated against data via model.matrix() to build the design matrix (with the usual factor expansion, interactions, and intercept; write ~ 0 + ... to drop the intercept).
constraints	A crr_constraints object. Ignored (with a warning) for method = "unconstrained".
data	Data frame used to evaluate X when it is a formula; ignored when X is already a matrix.
method	Sampler: "mhwg" (the paper's method) or "unconstrained" (baseline).
kernel	Dual kernel for "mhwg": "exponential" or "half_gaussian".
prior	A crr_prior object.
n_iter	Total MCMC iterations per chain (including warmup).
warmup	Warmup iterations discarded by summary(), plot(), and the as.mcmc/as_draws converters. All draws are stored.
thin	Thinning applied by the converters (draws are stored unthinned).
chains	Number of independent chains, run serially.
seed	Optional integer seed (set.seed() is called if supplied).
control	A crr_control object.
verbose	Print progress every 1000 iterations.

Value

An object of class crr_fit; see summary.crr_fit().

Examples

```
if (requireNamespace("lpSolve", quietly = TRUE)) {
  con <- crr_constraints(rbind(c(1, 1, 0), c(0, 1, 1)), b = c(1, 1))
  sim <- simulate_crr(n = 50, p = 2, constraints = con, seed = 1)
  fit <- crr(sim$Y, sim$X, con, n_iter = 200, warmup = 100,
             chains = 1, seed = 1)
  summary(fit)
}
```

`crr_benchmark` *Benchmark samplers on a common data set*

Description

Fits several methods to the same data and collects a comparison table: wall-clock time, MH acceptance, effective sample sizes and sampling efficiency (min ESS per second), in-sample fit, and — when the true coefficients are supplied, e.g. from `simulate_crr()` — estimation error (RMSE) and coverage of the central posterior intervals. This is the package's canonical way to compare the paper's constrained sampler with the unconstrained probit baseline (or any subset of registered methods) without writing bespoke benchmarking code.

Usage

```
crr_benchmark(
  Y,
  X,
  constraints,
  methods = c("mhwg", "unconstrained"),
  beta = NULL,
  prob = 0.95,
  ...
)
```

Arguments

`Y`	Response matrix (`n` x `d`).
`X`	Covariate matrix (`n` x `p`).
`constraints`	A crr_constraints object (used by constrained methods only).
`methods`	Character vector of methods accepted by `crr()`.
`beta`	True coefficient matrix (`p` x `d`), or `NULL`. When supplied, RMSE and interval coverage are reported.
`prob`	Central posterior interval probability for coverage.
`...`	Passed on to `crr()` (e.g. `n_iter`, `warmup`, `chains`, `kernel`, `seed`, `control`). The same arguments — including any seed — are used for every method.

Value

An object of class `crr_benchmark`: a list with `table` (one row per method) and `fits` (the underlying `crr_fit` objects, named by method).

Examples

```
if (requireNamespace("lpSolve", quietly = TRUE)) {
  con <- crr_constraints(rbind(c(1, 1, 0), c(0, 1, 1)), b = c(1, 1))
  sim <- simulate_crr(n = 50, p = 2, constraints = con, seed = 1)
  bm <- crr_benchmark(sim$Y, sim$X, con, beta = sim$beta,
                      n_iter = 200, warmup = 100, seed = 1)
  bm
}
```

---

crr_constraints *Constraint system for combinatorial responses*

Description

Defines the feasible set $\{y \in \{0,1\}^d : Ay \leq b\}$ of a combinatorial response. The matrix `A` must be integer-valued and, for the exactness of the dual-certificate augmentation used by `crr()`, totally unimodular.

Usage

```
crr_constraints(A, b, direction = "<=", check_tum = TRUE, dedup = TRUE)
```

Arguments

`A`	Integer constraint matrix (`m` x `d`).
`b`	Integer right-hand side vector of length `m`.
`direction`	Either a single string or a length-`m` character vector with entries `"<="` or `">="`. Internally all constraints are stored in `"<="` form.
`check_tum`	Verify total unimodularity of `A` (exhaustive check, exponential in `min(m, d)`). Set to `FALSE` for large matrices whose TUM structure is known, e.g. network/incidence matrices.
`dedup`	Remove redundant constraint rows: rows identical in `A` are collapsed to the one with the smallest `b`, since a tighter `A y <= b_1` implies any `A y <= b_2` with `b_2 >= b_1`. This leaves the feasible set unchanged, preserves total unimodularity, and drops redundant dual dimensions that otherwise slow the sampler. `TRUE` by default; set `FALSE` to keep the system exactly as supplied (e.g. to reproduce a simulation design with a fixed number of constraints).

Value

An object of class `crr_constraints` with elements `A`, `b` (in `<=` form), `m`, `d`.

Examples

```
A <- rbind(c(1, 1, 0), c(0, 1, 1))
con <- crr_constraints(A, b = c(1, 1))
```

`crr_control` *Sampler control parameters*

Description

Tuning parameters for the MH-within-Gibbs sampler in `crr()`.

Usage

```
crr_control(
  n_iter_hit_and_run = 50,
  rho = 1,
  zeta_block = "adaptive",
  zeta_target_accept = 0.6,
  max_dir_tries = 1000,
  bound_truncation = 1e+05,
  n_threads = 1
)
```

Arguments

`n_iter_hit_and_run`
: Inner hit-and-run steps per dual update. The default (50) balances dual mixing against per-sweep cost; smaller values are faster but mix the dual variable less well.

`rho`
: Rate of the exponential dual kernel (ignored for `kernel = "half_gaussian"`).

`zeta_block`
: Number of response coordinates updated per MH step for the latent utilities. Either a positive integer for a fixed block size (the paper uses `min(d, 100)`), or `"adaptive"` (the default) to tune the block size automatically during warmup to hit `zeta_target_accept`. The optimal block size depends on the constraint geometry, not just `d`, so the adaptive rule is more robust across problems; pass an integer to reproduce a fixed-block run.

`zeta_target_accept`
: Target MH acceptance rate for the adaptive block controller (ignored when `zeta_block` is an integer). The default `0.6` maximizes mixing efficiency across a wide range of problems.

`max_dir_tries`
: Maximum attempts to draw a valid hit-and-run direction.

`bound_truncation`
: Truncation applied to infinite line-segment bounds inside hit-and-run.

`n_threads`
: OpenMP threads for the per-observation dual updates. Results do not depend on this value (each observation has its own RNG stream).

Value

An object of class `crr_control`.

`crr_diagnostics` *MCMC and regression diagnostics report*

Description

Assembles a structured diagnostics report for a fitted model: per-parameter posterior summaries with effective sample sizes, sampling efficiency (ESS per second), split-Rhat convergence diagnostics, MH acceptance rates, in-sample regression fit (exact-match and per-coordinate accuracy of the fitted responses), and posterior predictive goodness-of-fit checks that calibrate the fit statistics against what the model itself predicts (see `crr_ppc()`). When the true coefficients are known (simulation), supplying `beta` adds estimation error (RMSE) and interval coverage. Potential problems (low ESS, high Rhat, low acceptance, extreme predictive p-values) are collected as warnings and flagged by `print()`.

Usage

```
crr_diagnostics(object, beta = NULL, prob = 0.95, n_rep = 50)
```

Arguments

`object`	A `crr_fit` object.
`beta`	Optional true coefficient matrix (p x d), e.g. from `simulate_crr()`.
`prob`	Central posterior interval probability for the summary table.
`n_rep`	Posterior predictive replicates for the goodness-of-fit checks; set to `0` to skip them.

Value

An object of class `crr_diagnostics`: a list with elements `table` (per-parameter data frame with columns `mean`, `sd`, interval bounds, `ess`, `ess_per_sec`, `rhat`), `fit_stats` (exact-match rate and per-coordinate accuracy, or `NULL` if fitted responses are unavailable), `ppc` (a crr_ppc object, or `NULL` if skipped/unavailable), `truth` (RMSE, coverage, and the error matrix `coef(object) - beta`, or `NULL` if `beta` was not supplied), `warnings` (character vector), and the sampler configuration.

Examples

```
if (requireNamespace("lpSolve", quietly = TRUE)) {
  con <- crr_constraints(rbind(c(1, 1, 0), c(0, 1, 1)), b = c(1, 1))
  sim <- simulate_crr(n = 50, p = 2, constraints = con, seed = 1)
  fit <- crr(sim$Y, sim$X, con, n_iter = 200, warmup = 100, seed = 1)
  crr_diagnostics(fit, beta = sim$beta, n_rep = 20)
}
```

crr_ess *Effective sample sizes*

Description

Effective sample size per coefficient, computed from post-warmup thinned draws and summed across chains (via `coda::effectiveSize()`).

Usage

```
crr_ess(object)
```

Arguments

`object`	A `crr_fit` object.

Value

Named numeric vector of length `p * d`.

Examples

```
if (requireNamespace("lpSolve", quietly = TRUE)) {
  con <- crr_constraints(rbind(c(1, 1, 0), c(0, 1, 1)), b = c(1, 1))
  sim <- simulate_crr(n = 50, p = 2, constraints = con, seed = 1)
  fit <- crr(sim$Y, sim$X, con, n_iter = 200, warmup = 100, seed = 1)
  crr_ess(fit)   # effective sample size per coefficient; larger means better
                 # mixing (a few hundred is usually plenty for point estimates)
}
```

crr_ppc *Posterior predictive goodness-of-fit checks*

Description

Simulates replicated response matrices from the posterior predictive distribution (draw β from the retained draws, draw $\zeta = X\beta + \varepsilon$, map each row to its feasible maximizer) and compares observed test statistics against their predictive distribution. Statistics: the exact-match rate and per-coordinate accuracy of the point predictor `fitted(object)` applied to each replicate, and the per-coordinate marginal response frequencies.

Usage

```
crr_ppc(object, n_rep = 50, seed = NULL)
```

Arguments

object	A crr_fit object.
n_rep	Number of posterior predictive replicates.
seed	Optional integer seed.

Details

Point-prediction accuracy is bounded by the intrinsic utility noise, so a low raw exact-match rate does not by itself indicate misfit: the model fits well when the *observed* statistics are typical of the predictive distribution (two-sided p-values away from 0). Constrained fits require the 'lpSolve' package.

Value

An object of class crr_ppc: a list with observed (named statistic vector), replicated (n_rep x n_stats matrix), p_value (two-sided predictive p-values per statistic), and n_rep.

Examples

```
if (requireNamespace("lpSolve", quietly = TRUE)) {
  con <- crr_constraints(rbind(c(1, 1, 0), c(0, 1, 1)), b = c(1, 1))
  sim <- simulate_crr(n = 50, p = 2, constraints = con, seed = 1)
  fit <- crr(sim$Y, sim$X, con, n_iter = 200, warmup = 100, seed = 1)
  crr_ppc(fit, n_rep = 20, seed = 1)
}
```

---

crr_prior *Prior specification for crr models*

Description

Currently a single family: independent Gaussian priors $\beta_{kj} \sim N(0, sd^2)$ on all regression coefficients.

Usage

```
crr_prior(family = "gaussian", sd = 1)
```

Arguments

family	Prior family; only "gaussian" is implemented.
sd	Prior standard deviation (scalar, positive).

Value

An object of class crr_prior.

Examples

```
crr_prior()               # N(0, 1)
crr_prior(sd = 10)        # weakly informative
```

`crr_rhat` *Split-Rhat convergence diagnostics*

Description

Potential scale reduction factor per coefficient, computed on split half-chains so it is defined even for `chains = 1`.

Usage

```
crr_rhat(object)
```

Arguments

`object`	A `crr_fit` object.

Value

Named numeric vector of length `p * d`.

Examples

```
if (requireNamespace("lpSolve", quietly = TRUE)) {
  con <- crr_constraints(rbind(c(1, 1, 0), c(0, 1, 1)), b = c(1, 1))
  sim <- simulate_crr(n = 50, p = 2, constraints = con, seed = 1)
  fit <- crr(sim$Y, sim$X, con, n_iter = 200, warmup = 100, seed = 1)
  crr_rhat(fit)  # values near 1 indicate convergence; above ~1.05 is a
                 # sign the chain needs to run longer
}
```

`draw_utility` *Draw latent utilities from their truncated-normal full conditional*

Description

For each coordinate `j` in `subset`, draws $\zeta_{\cdot j} \sim TN(\mu_{\cdot j}, 1)$ truncated below at $(UA)_{\cdot j}$ where $y = 1$ and above where $y = 0$.

Usage

```
draw_utility(zeta, Mu, Y, UA, subset = seq_len(ncol(zeta)))
```

Arguments

zeta	Current latent utility matrix (n x d); returned unchanged outside subset.
Mu	Mean matrix (n x d), e.g. X %*% beta.
Y	Response matrix (n x d).
UA	Dual bounds U %*% A (n x d).
subset	Integer vector of coordinates to update.

Value

Matrix (n x d).

---

dual_feasible *Check dual-certificate feasibility*

Description

Verifies, per observation, that a dual matrix lies in $\mathcal{D}(y_i, \zeta_i) \cap \{u \geq 0\}$.

Usage

```
dual_feasible(constraints, zeta, Y, U, control = crr_control())
```

Arguments

constraints	A crr_constraints object.
zeta	Latent utility matrix (n x d).
Y	Response matrix (n x d).
U	Current dual matrix (n x m); must be feasible.
control	A crr_control object.

Value

Integer 0/1 vector of length n.

fitted.crr_fit *Fitted responses*

Description

Posterior-point-estimate responses on the training covariates: `predict(object, type = "response")`.

Usage

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

Arguments

`object`	A `crr_fit` object.
`...`	Unused.

Value

Binary matrix (`n` x `d`).

init_dual *Initialize dual certificates*

Description

Computes the active-constraint mask `active` ($1\{Ay_i = b\}$) and an initial dual matrix `U` with independent Exp(1) entries on the active support.

Usage

```
init_dual(constraints, Y)
```

Arguments

`constraints`	A crr_constraints object.
`Y`	Response matrix (`n` x `d`).

Value

List with `U` (`n` x `m`) and `active` (`n` x `m` binary mask).

is_feasible *Check feasibility of responses*

Description

Check feasibility of responses

Usage

```
is_feasible(constraints, Y)
```

Arguments

constraints	A crr_constraints object.
Y	Response matrix (n x d), rows are responses.

Value

Logical vector of length n: does each row satisfy `A y <= b` and y in {0,1}?

is_tum *Check total unimodularity*

Description

Exhaustively checks whether every square submatrix of A has determinant in {-1, 0, 1}. Cost grows exponentially in `min(nrow(A), ncol(A))`; use only for small-to-moderate matrices.

Usage

```
is_tum(A)
```

Arguments

A	Integer matrix.

Value

TRUE or FALSE, with attribute "witness" (a list with rows, cols, determinant) when FALSE.

Examples

```
is_tum(rbind(c(1, 1, 0), c(0, 1, 1)))       # TRUE
is_tum(rbind(c(1, 1, 0), c(1, 0, 1), c(0, 1, 1)))  # FALSE
```

`plot.crr_fit` *Diagnostic plots for a crr fit*

Description

All plots use post-warmup thinned draws.

Usage

```
## S3 method for class 'crr_fit'
plot(
  x,
  pars = NULL,
  type = c("trace", "acf", "violin", "ess", "ess_time", "residual", "beta_diff"),
  beta = NULL,
  ...
)
```

Arguments

`x`	A `crr_fit` object.
`pars`	Character vector of parameter names (e.g. `"beta[1,2]"`). Defaults to the first four parameters for `"trace"`/`"acf"` and all parameters otherwise. Ignored for `"residual"` and `"beta_diff"`.
`type`	Plot type; see Details.
`beta`	True coefficient matrix (p x d); required for `type = "beta_diff"`.
`...`	Passed to the underlying base plotting functions.

Details

- `"trace"`: per-parameter trace plots, chains overlaid.
- `"acf"`: per-parameter autocorrelation functions (chains pooled).
- `"violin"`: violin plot of the marginal posterior of each coefficient, with median and central 95% interval.
- `"ess"`: effective sample size per coefficient, with a dashed reference line at 100.
- `"ess_time"`: effective sample size per second of sampling time — the efficiency measure used for method comparison in the paper.
- `"residual"`: heat map of the training residuals `Y - fitted(x)` (entries in {-1, 0, 1}); rows are observations, columns response coordinates.
- `"beta_diff"`: heat map of the coefficient estimation error `coef(x) - beta` against a known true coefficient matrix (supplied via `beta`, e.g. from `simulate_crr()`), with the error printed in each cell.

Value

`x`, invisibly.

predict.crr_fit *Posterior predictions*

Description

Posterior predictions

Usage

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

Arguments

object	A crr_fit object.
newdata	Covariate matrix (n_new x p); defaults to the training covariates.
type	"utility" returns posterior-mean latent utilities $X\hat{\beta}$; "response" additionally maps each utility row to a feasible response — the constrained maximizer via integer programming for constrained fits (requires the 'lpSolve' package), or the coordinatewise sign indicator $1\{x^\top \hat{\beta} > 0\}$ for method = "unconstrained" fits.
...	Unused.

Value

Matrix (n_new x d) of utilities or feasible responses.

Examples

```
if (requireNamespace("lpSolve", quietly = TRUE)) {
  con <- crr_constraints(rbind(c(1, 1, 0), c(0, 1, 1)), b = c(1, 1))
  sim <- simulate_crr(n = 50, p = 2, constraints = con, seed = 1)
  fit <- crr(sim$Y, sim$X, con, n_iter = 200, warmup = 100, seed = 1)
  head(predict(fit, type = "response"))  # feasible 0/1 predictions; each row
                                         # satisfies the constraints A y <= b
}
```

random_constraints *Generate a random totally unimodular constraint system*

Description

For small problems (d * m <= 50 and m >= 2 fails), draws random {-1, 0, 1} matrices until one is totally unimodular, with b = 1. For larger problems, generates a random network (incidence-type) matrix with one +1 and one -1 per row — totally unimodular by construction — and Bernoulli right-hand sides. This mirrors the simulation design of the paper.

Usage

```
random_constraints(d, m)
```

Arguments

d	Response dimension.
m	Number of constraints.

Value

A crr_constraints object.

`residuals.crr_fit` *Response residuals*

Description

Training residuals `Y - fitted(object)`, entries in {-1, 0, 1}.

Usage

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

Arguments

object	A `crr_fit` object.
...	Unused.

Value

Integer matrix (`n` x `d`).

sample_dual *Update dual certificates given latent utilities*

Description

One Gibbs update of $U \mid \zeta, y$: for each observation, runs hit-and-run over the dual-certificate polytope $\mathcal{D}(y_i, \zeta_i) \cap \{u \geq 0\}$ targeting the chosen kernel, and returns the endpoint.

Usage

```
sample_dual(
  constraints,
  zeta,
  Y,
  U,
  active,
  kernel = "exponential",
  control = crr_control()
)
```

Arguments

constraints	A crr_constraints object.
zeta	Latent utility matrix (n x d).
Y	Response matrix (n x d).
U	Current dual matrix (n x m); must be feasible.
active	Active-constraint mask from init_dual().
kernel	"exponential" or "half_gaussian".
control	A crr_control object.

Value

Updated dual matrix (n x m).

sample_utility *Metropolis-Hastings update of the latent utilities*

Description

One MH step of $\zeta \mid U, \beta, y$: proposes new utilities for a random block of coordinates from the truncated-normal proposal, forms the coordinatewise envelope $\tilde{\zeta}$ (max where $y = 1$, min where $y = 0$), runs hit-and-run under $\tilde{\zeta}$ to obtain a dual certificate $U^\star$, and accepts each observation's proposal iff its $u_i^\star$ is feasible for the current ζ_i.

Usage

```
sample_utility(
  constraints,
  zeta,
  Y,
  U,
  active,
  Mu,
  kernel = "exponential",
  control = crr_control(),
  block = NULL
)
```

Arguments

constraints	A crr_constraints object.
zeta	Latent utility matrix (n x d).
Y	Response matrix (n x d).
U	Current dual matrix (n x m); must be feasible.
active	Active-constraint mask from init_dual().
Mu	Mean matrix (n x d), e.g. X %*% beta.
kernel	"exponential" or "half_gaussian".
control	A crr_control object.
block	Number of coordinates to update this step. When NULL (default), it is resolved from control$zeta_block: the fixed integer if set, otherwise min(d, 100) (the adaptive controller in crr() supplies an explicit value each step, so this fallback only applies to direct callers under an adaptive control).

Value

List with zeta (updated matrix) and accept (0/1 vector, per-observation acceptance).

---

simulate_crr *Simulate combinatorial response regression data*

Description

Draws $X_{ik} \sim N(0,1)$, coefficients $\beta_{kj} \sim N(0,1)$ (unless supplied), latent utilities $\zeta_i = \beta^\top x_i + \varepsilon_i$ with standard normal noise, and responses $y_i = \arg\max_{y \in \mathcal{Y}} \zeta_i^\top y$ solved by integer programming.

Usage

```
simulate_crr(n, p, constraints, beta = NULL, seed = NULL)
```

Arguments

n	Number of observations.
p	Number of covariates.
constraints	A crr_constraints object defining $\mathcal{Y}$.
beta	Optional true coefficient matrix (p x d).
seed	Optional integer seed.

Value

List with Y (n x d), X (n x p), beta (p x d, the truth), zeta (n x d), and constraints.

Examples

```
if (requireNamespace("lpSolve", quietly = TRUE)) {
  con <- crr_constraints(rbind(c(1, 1, 0), c(0, 1, 1)), b = c(1, 1))
  sim <- simulate_crr(n = 20, p = 2, constraints = con, seed = 1)
}
```

---

summary.crr_fit *Posterior summary of a crr fit*

Description

Posterior summary of a crr fit

Usage

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

Arguments

object	A crr_fit object.
...	Unused.

Value

A data frame with posterior mean, sd, and 2.5/50/97.5 percent quantiles per coefficient, computed from post-warmup thinned draws pooled across chains.

`update_coef` *Conjugate Gaussian update of the regression coefficients*

Description

Draws $\beta_{\cdot j} \sim N(V X^\top \zeta_{\cdot j}, V)$ jointly for all response coordinates.

Usage

```
update_coef(X, zeta, precomp)
```

Arguments

`X`	Covariate matrix (`n` x `p`).
`zeta`	Latent utility matrix (`n` x `d`).
`precomp`	Output of `coef_precompute()`.

Value

Coefficient matrix (`p` x `d`).

Index