Package 'cevcmm'

July 24, 2026

Title Communication-Efficient Varying Coefficient Mixed-Effects Models

Version 0.1.3

Description Scalable inference for Varying Coefficient Mixed-Effects Models (VCMMs) with large, correlated random effects. Implements sufficient-statistics, one-step communication-efficient surrogate likelihood, and SVD-stabilized (Singular Value Decomposition) estimators with Kronecker and separable covariance structures. Implements the methodology of Jalili and Lin (2025) <doi:10.48550/arXiv.2511.12732>.

License GPL (>= 3)

Encoding UTF-8

VignetteBuilder knitr

URL https://lidajalili.github.io/cevcmm/, https://github.com/lidajalili/cevcmm

BugReports https://github.com/lidajalili/cevcmm/issues

Depends R (>= 4.0.0)

Imports splines, Rcpp (>= 1.0.0)

LinkingTo Rcpp, RcppArmadillo

Suggests devtools, microbenchmark, RSpectra, testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Lida Jalili [aut, cre], Li-Hsiang Lin [aut]

Maintainer Lida Jalili <lchalangarjalilideh1@gsu.edu>

Repository CRAN

Date/Publication 2026-07-24 09:20:09 UTC

Contents

`+.vcmm_ss` *Additive aggregation of sufficient-statistics summaries*

Description

Defines the `+` method for `vcmm_ss` objects, so that `Reduce("+", summaries)` aggregates a list of node summaries element-wise across the six sufficient-statistics components.

Usage

```
## S3 method for class 'vcmm_ss'
e1 + e2
```

Arguments

`e1, e2`	`vcmm_ss` objects.

Details

Dimensions (p, q) must match between operands; an informative error is thrown otherwise.

Value

A `vcmm_ss` object holding the component-wise sums.

`accumulate_stats` *Add one batch or node statistics to a running accumulator*

Description

Performs the additive aggregation `acc <- acc + stats` component by component. After processing all batches, the accumulator holds the full-data sufficient summary.

Usage

```
accumulate_stats(acc, stats)
```

Arguments

`acc`	A `vcmm_accumulator` object from `init_accumulator()`.
`stats`	A `vcmm_ss` object from `compute_sufficient_stats()`.

Value

The updated accumulator (class `"vcmm_accumulator"`).

See Also

Other sufficient statistics: `compute_sufficient_stats()`, `init_accumulator()`

Examples

```
set.seed(1)
n_batch <- 50; p <- 3; q <- 2
acc <- init_accumulator(p, q)

for (b in 1:3) {
  X <- cbind(1, matrix(rnorm(n_batch * (p - 1)), n_batch, p - 1))
  Z <- matrix(rnorm(n_batch * q), n_batch, q)
  y <- rnorm(n_batch)
  ss <- compute_sufficient_stats(y, X, Z)
  acc <- accumulate_stats(acc, ss)
}

acc$n_obs  # 150
```

build_kronecker_precision *Build the Kronecker-structured random-effect precision matrix*

Description

For $\Sigma_\alpha = \Sigma_{\text{left}} \otimes \Sigma_{\text{right}}$ (column-stacking convention), returns

$$\sigma_\varepsilon^2 \cdot (\Sigma_{\text{left}}^{-1} \otimes \Sigma_{\text{right}}^{-1})$$

which is added to `crossprod(Z)` inside the random-effect block of the VCMM Hessian. This is the structured analogue of `(sigma_eps^2 / sigma_alpha^2) * I_q` used under `re_cov = "diag"`.

Usage

```
build_kronecker_precision(Sigma_left, Sigma_right, sigma_eps)
```

Arguments

`Sigma_left`	A $k \times k$ positive-definite numeric matrix ($k = 2$ for OD-style; arbitrary k for general separable).
`Sigma_right`	A $G \times G$ positive-definite numeric matrix.
`sigma_eps`	Positive numeric. Residual standard deviation.

Value

A $kG \times kG$ numeric matrix.

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

build_penalty_matrix *Build a B-spline second-order difference penalty matrix*

Description

Constructs a symmetric $p \times p$ penalty matrix $\mathbf{P}_\lambda$ for use with penalised B-spline varying coefficients. For a single varying coefficient (`n_blocks = 1`, default), $p = n\_basis + 1$; the intercept (row/column 1) is unpenalised, and the remaining block is $\lambda \mathbf{D}_2^\top \mathbf{D}_2$ where $\mathbf{D}_2$ is the second-order difference operator on `n_basis` coefficients.

Usage

```
build_penalty_matrix(n_basis, lambda, n_blocks = 1L)
```

Arguments

n_basis	Integer (≥ 3). Number of B-spline basis columns per varying coefficient.
lambda	Non-negative numeric. Smoothing parameter λ.
n_blocks	Integer (≥ 1). Number of varying coefficients (covariates) sharing the same penalty structure. Default 1.

Details

For n_blocks > 1 (multiple covariates each with their own varying coefficient), the result is block-diagonal: one zero entry for the intercept, followed by n_blocks copies of the same n_basis x n_basis penalty block. The total dimension is 1 + n_blocks * n_basis.

A small ridge is added if any block is not positive-definite, to guarantee numerical stability in downstream solves.

Value

A symmetric (1 + n_blocks * n_basis) x (1 + n_blocks * n_basis) numeric matrix. Row/column 1 is zero (intercept is unpenalised).

References

Eilers, P. H. C. and Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. *Statistical Science*, 11(2), 89–121.

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

Examples

```
# Single varying coefficient
P1 <- build_penalty_matrix(n_basis = 10, lambda = 1)
dim(P1)            # 11 x 11
P1[1, 1]           # 0 -- intercept is not penalised

# Three varying coefficients sharing the same penalty
P3 <- build_penalty_matrix(n_basis = 10, lambda = 1, n_blocks = 3)
dim(P3)            # 31 x 31  (1 + 3*10)
isSymmetric(P3)
```

---

build_vcmm_design *Build the design matrix and penalty for a VCMM*

Description

Constructs the fixed-effects design matrix and matching penalty for a VCMM with one or more varying coefficients. Given covariates `X` (n by K) and a time/index vector `t` (length n), builds the cubic-by-default B-spline basis `B` at `t` and assembles

- `X_design` = `cbind(1, B * X[,1], B * X[,2], ..., B * X[,K])`, dimension n by (1 + K * n_basis); the first column is the intercept, then each covariate gets its own n_basis-column block.
- `penalty`: a (1 + K * n_basis) by (1 + K * n_basis) block-diagonal second-order difference penalty matrix with the intercept unpenalised.

This is the natural multi-covariate generalisation of the single-covariate design used in the simulation code, matching the paper's general VCMM specification.

Usage

```
build_vcmm_design(
  X,
  t,
  n_basis = NULL,
  degree = 3L,
  lambda = 1,
  normalize_t = TRUE
)
```

Arguments

`X`	Numeric n by K matrix (or length-n vector if K = 1) of covariates that get varying coefficients in `t`.
`t`	Numeric vector of length n. The variable in which the coefficients vary smoothly (time, index, location, etc.).
`n_basis`	Integer ($\geq$ `degree` + 1) or `NULL`. Number of B-spline basis columns per varying coefficient. If `NULL` (default), chosen as `max(floor(n^(1/3)) + 4, 10)` matching the simulation code's rule of thumb.
`degree`	Integer. B-spline degree (default 3 = cubic).
`lambda`	Non-negative numeric. Smoothing parameter for the penalty (default 1).
`normalize_t`	Logical. If `TRUE` (default), `t` is linearly mapped to `[0, 1]` before building the basis. Set `FALSE` only if `t` is already in `[0, 1]`.

Value

A list with elements:

- `X_design`: numeric n by (1 + K * n_basis) design matrix.
- `penalty`: numeric (1 + K * n_basis) by (1 + K * n_basis) penalty matrix.
- `B_spline`: numeric n by n_basis basis matrix at `t`.
- `internal_knots`, `boundary_knots`: spline knot locations, recorded so predictions at new `t` can use the same basis.

- `degree`, `n_basis`, `K`, `lambda`: scalars recording how the design was built.
- `normalize_t`, `t_min`, `t_max`: how to remap new `t` values at prediction time.

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

Examples

```
set.seed(1)
n <- 200
t <- runif(n)
x1 <- runif(n); x2 <- runif(n)

# Single varying coefficient
d1 <- build_vcmm_design(X = x1, t = t, n_basis = 8)
dim(d1$X_design)   # 200 x 9   (1 + 1*8)

# Two varying coefficients
d2 <- build_vcmm_design(X = cbind(x1, x2), t = t, n_basis = 8)
dim(d2$X_design)   # 200 x 17  (1 + 2*8)
```

---

coef.vcmm_fit *Fixed-effects coefficient vector from a vcmm fit*

Description

Returns $\hat{\beta}$ as a named numeric vector. Under the package's design (`X_design = cbind(1, B*x_1, ..., B*x_K)`, see `build_vcmm_design`), entry 1 is the constant intercept and entries `(1 + (k-1)*m + 1):(1 + k*m)` are the spline basis coefficients for the k-th varying coefficient $\beta_k(t)$.

Usage

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

Arguments

`object`	A `vcmm_fit` object.
`...`	Unused.

Details

To evaluate $\beta_k(t)$ at user-supplied `t` values, use `varying_coef`. To get the same vector reshaped into a (basis x covariate) matrix with the intercept split out, use `fixef.vcmm_fit`.

Value

Named numeric vector of length $p = 1 + K \cdot m$.

`compute_sufficient_stats` *Compute one batch or node sufficient statistics for a normal linear VCMM*

Description

Computes the six-component summary that fully encodes one node's contribution to the normal linear VCMM likelihood. These statistics are additive across nodes and batches, so a central server can recover the full-data estimator by summing per-node summaries – without ever seeing the raw response or design matrices.

Usage

```
compute_sufficient_stats(y, X, Z, use_cpp = TRUE)
```

Arguments

`y`	Numeric response vector of length n.
`X`	Numeric n by p fixed-effects design matrix (intercept plus spline basis columns).
`Z`	Numeric n by q random-effects design matrix.
`use_cpp`	Logical. If `TRUE` (default), dispatch to the RcppArmadillo backend `compute_sufficient_stats_cpp()`. If `FALSE`, use the pure-R `crossprod()` reference path. The two paths agree to within floating-point summation order (typically below 1e-13 relative).

Details

The returned components are:

- `a`: `sum(y^2)`, a scalar.
- `b`: `crossprod(X, y)`, dimension p by 1.
- `C`: `crossprod(X)`, dimension p by p.
- `ZtZ`: `crossprod(Z)`, dimension q by q.
- `Zty`: `crossprod(Z, y)`, dimension q by 1.
- `XtZ`: `crossprod(X, Z)`, dimension p by q.

Since Day 16 the default backend is a RcppArmadillo implementation (`use_cpp = TRUE`). Pass `use_cpp = FALSE` to fall back to the pure-R reference path; this is used by the Day-16 bit-equivalence validation and is also useful for debugging.

Value

A list of class `"vcmm_ss"` with elements `a`, `b`, `C`, `ZtZ`, `Zty`, `XtZ`, and `n_obs` (the number of observations summarized).

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

See Also

Other sufficient statistics: `accumulate_stats()`, `init_accumulator()`

Examples

```
set.seed(1)
n <- 100; p <- 3; q <- 2
X <- cbind(1, matrix(rnorm(n * (p - 1)), n, p - 1))
Z <- matrix(rnorm(n * q), n, q)
y <- rnorm(n)

ss <- compute_sufficient_stats(y, X, Z)
str(ss)
```

---

estimate_kronecker_components

Moment-based estimator for the Kronecker left component

Description

Given a length-kG random-effects estimate $\hat{\alpha} = \mathrm{vec}_{\mathrm{col}}(M)$ (column-stacked, $M \in \mathbb{R}^{G \times k}$) and the right-side covariance `Sigma_right`, returns the weighted moment estimate

$$\hat{\Sigma}_{\mathrm{left}} = \tfrac{1}{G}\, M^{\top}\, \Sigma_{\mathrm{right}}^{-1}\, M.$$

Usage

```
estimate_kronecker_components(alpha, n_groups, q_left = 2L, Sigma_right = NULL)
```

Arguments

`alpha`	Numeric vector of length kG.
`n_groups`	Integer G.
`q_left`	Integer k, the left (within) dimension. Defaults to 2 for backward compatibility with the OD setting.
`Sigma_right`	Optional $G \times G$ positive-definite covariance. If `NULL`, the unweighted sample covariance `cov(alpha_mat)` is returned (unbiased only when rows are uncorrelated).

Details

This is unbiased under the Kronecker model $\alpha \sim N(0, \Sigma_{\text{left}} \otimes \Sigma_{\text{right}})$ when `Sigma_right` is correct. When $\hat{\alpha}$ is the BLUP rather than the true α, apply the EM-style correction in `vcmm` (handled automatically by `fit_ss` / `fit_csl`).

Backwards-compatible alias: if `q_left = 2` (the default), this is the same estimator as the previous `estimate_kronecker_components` for the OD setting.

Value

A $k \times k$ symmetric positive-definite matrix.

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

`fit_csl` *Fit a VCMM via the one-step CSL estimator*

Description

Performs a single Newton refinement of a pilot estimator using the aggregated sufficient statistics, implementing the one-step communication-efficient surrogate-likelihood (CSL) estimator of Jalili and Lin (2025, Section 3). For the normal linear VCMM the update is

$$\widehat{\theta}_{CSL} = \widehat{\theta}_0 - K^{-1}\, g(\widehat{\theta}_0),$$

where $\theta = (\beta, \alpha)$, the gradient g uses the full aggregated stats, and the Hessian K also uses the full aggregated stats with prior augmentation. Theorem 3.1 of the paper shows that whenever the pilot estimator is $\sqrt{N}$-consistent, the one-step CSL estimator is first-order equivalent to the full SS estimator.

Usage

```
fit_csl(
  stats,
  penalty,
  control = vcmm_control(),
  pilot = NULL,
  pilot_max_iter = 5L,
  pilot_tol_beta = 0.001,
  pilot_tol_alpha = 0.001,
  re_cov_state = NULL
)
```

Arguments

stats	A vcmm_ss or vcmm_accumulator object containing the aggregated sufficient statistics.
penalty	A symmetric $p \times p$ penalty matrix from build_penalty_matrix().
control	A vcmm_control object with fitting options. The sigma_eps and sigma_alpha entries provide the initial variance values used by the internal pilot (when pilot = NULL).
pilot	Optional vcmm_fit object to use as the pilot estimator. If NULL (default), an internal SS pilot is run.
pilot_max_iter	Integer. Maximum iterations for the internal SS pilot (default 5). Ignored if pilot is supplied.
pilot_tol_beta	Positive numeric. Loose tolerance for the internal SS pilot (default 1e-3). Ignored if pilot is supplied.
pilot_tol_alpha	Positive numeric. Loose tolerance for the internal SS pilot (default 1e-3). Ignored if pilot is supplied.
re_cov_state	Optional. An internal random-effects covariance state object (NULL for diagonal, or constructed for kronecker via vcmm() with re_cov = "kronecker"). Passed through to the internal SS pilot and reused for the Newton-step Hessian. Advanced users typically reach re_cov_state via vcmm() rather than calling fit_csl() directly.

Details

Pilot estimator. If pilot = NULL (default), an internal SS pilot is run with loose tolerances and a small number of iterations (pilot_max_iter = 5, pilot_tol_beta = 1e-3, pilot_tol_alpha = 1e-3); these defaults match the dense simulation study of the paper. Setting pilot_max_iter = 1L gives the cheapest possible pilot (the OLS-like single-step pilot used in the origin-destination simulation), still $\sqrt{N}$-consistent in the normal linear case. Advanced users may pass any vcmm_fit object via the pilot argument – e.g. a previously fitted fit_ss() result.

Hessian. This implementation uses the full aggregated Hessian, not the reference-node curvature approximation $\tilde{K}$ from the paper. The two are first-order equivalent under the conditions of Theorem 3.1; the full-aggregated form is the most accurate and is the natural default for a single-server fit, which is the typical use case of this package.

Value

A list of class "vcmm_fit" with the same fields as fit_ss(), plus:

- pilot: the vcmm_fit pilot used.
- pilot_elapsed_sec: pilot fitting time.
- step_elapsed_sec: Newton step time.

The method field is "CSL".

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

Examples

```
set.seed(1)
n <- 500; p <- 4; q <- 3
X <- cbind(1, matrix(rnorm(n * (p - 1)), n, p - 1))
Z <- matrix(rnorm(n * q), n, q)
alpha_true <- rnorm(q, sd = 0.5)
y <- as.vector(
  X %*% c(2, 0.5, -0.3, 0.8) + Z %*% alpha_true + rnorm(n, sd = 0.5)
)

stats <- compute_sufficient_stats(y, X, Z)
P     <- build_penalty_matrix(n_basis = p - 1, lambda = 0.1)

# Default: internal SS pilot (5 loose iterations) then one Newton step
fit <- fit_csl(stats, P,
               vcmm_control(sigma_eps = 0.5, sigma_alpha = 0.5))
fit

# Cheapest pilot: single SS step
fit_one <- fit_csl(stats, P,
                   vcmm_control(sigma_eps = 0.5, sigma_alpha = 0.5),
                   pilot_max_iter = 1L)

# Advanced: user-supplied pilot
my_pilot <- fit_ss(stats, P,
                   vcmm_control(sigma_eps = 0.5, sigma_alpha = 0.5,
                                max_iter = 3))
fit_user <- fit_csl(stats, P,
                    vcmm_control(sigma_eps = 0.5, sigma_alpha = 0.5),
                    pilot = my_pilot)
```

`fit_from_summaries` *Fit a VCMM from aggregated sufficient-statistics summaries*

Description

Server-side fit using only the small per-node summaries; no raw data required. The mathematical guarantee (Theorem 1 of Jalili and Lin, 2025) is that for a fixed partition of the data into nodes, the fit obtained here is identical to the one a centralised `vcmm()` call would produce on the pooled data, up to floating-point summation noise.

Usage

```
fit_from_summaries(
  summaries,
  penalty,
  control = vcmm_control(),
  method = c("csl", "ss"),
  re_cov = c("diag", "kronecker", "separable"),
  n_groups = NULL,
  q_left = NULL,
  Sigma_left_init = NULL,
  Sigma_right_init = NULL,
  Sigma_2x2_init = NULL,
  Sigma_spatial_init = NULL,
  Sigma_q_init = NULL,
  Omega_G_init = NULL,
  rowsum_constant = NULL,
  ...
)
```

Arguments

summaries	Either a list of vcmm_ss objects (one per node, the typical case) OR a single vcmm_ss that has already been aggregated (e.g., via Reduce("+", ...)) OR a vcmm_accumulator.
penalty	The $p \times p$ smoothing-penalty matrix used in the spline basis the nodes shared. Build it via the package's build_vcmm_design() (returned as design$penalty) or directly via build_penalty_matrix().
control	A vcmm_control object.
method	Either "csl" (default) or "ss".
re_cov	Either "diag", "kronecker", or "separable". See vcmm for full semantics.
n_groups	Required if re_cov is "kronecker" or "separable".
q_left	Required for "separable"; defaults to 2 for "kronecker".
Sigma_left_init, Sigma_right_init, Sigma_2x2_init, Sigma_spatial_init, Sigma_q_init, Omega_G_init	Same aliases accepted as in vcmm().
rowsum_constant	Optional numeric. If supplied and non-zero, apply the same identifiability re-centering that vcmm() applies post-fit. See Details.
...	Passed to fit_csl.

Details

Identifiability re-centering. vcmm() applies a post-fit re-centering when rowSums(Z) is constant (indicator-Z OD designs and similar), absorbing rowsum_constant * mean(alpha_hat) into beta_0. The server has no access to Z, so the caller must signal that the original Z had constant row sums by passing rowsum_constant. Default NULL means "no re-centering" (appropriate for dense-Z and block-Z designs).

Value

A vcmm_fit object.

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

See Also

node_summary, vcmm.

Examples

```
set.seed(1)
n <- 600
t  <- runif(n); x <- runif(n); Z <- matrix(rnorm(n * 4), n, 4)
a_true <- rnorm(4, sd = 0.3)
y <- 2 + sin(2 * pi * t) * x + as.vector(Z %*% a_true) + rnorm(n, sd = 0.5)

design <- build_vcmm_design(X = x, t = t)
Xd     <- design$X_design
ctrl   <- vcmm_control(sigma_eps = 0.5, sigma_alpha = 0.3)

# Split into 3 nodes and aggregate
idx_node <- sample.int(3, n, replace = TRUE)
summaries <- lapply(seq_len(3), function(s) {
  ii <- which(idx_node == s)
  node_summary(y[ii], Xd[ii, , drop = FALSE], Z[ii, , drop = FALSE])
})
fit <- fit_from_summaries(summaries,
                          penalty = design$penalty,
                          control = ctrl)
```

---

fit_ss *Fit a VCMM via the sufficient-statistics estimator*

Description

Iteratively solves for the fixed-effects coefficient vector beta and the random-effects vector alpha using only the aggregated sufficient summary, implementing Algorithm 1 of Jalili and Lin (2025) for the normal linear VCMM. The raw response and design matrices are not needed: everything reads off stats, which may come from a single call to compute_sufficient_stats() or from a streaming accumulator (init_accumulator() plus repeated accumulate_stats()).

Usage

```
fit_ss(stats, penalty, control = vcmm_control(), re_cov_state = NULL)

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

Arguments

stats	A vcmm_ss or vcmm_accumulator object containing the aggregated sufficient statistics.
penalty	A symmetric $p \times p$ penalty matrix from build_penalty_matrix().
control	A vcmm_control object with fitting options. Pass vcmm_control() to use defaults.
re_cov_state	Optional. An internal random-effects covariance state object (NULL for diagonal, or constructed for kronecker via vcmm() with re_cov = "kronecker"). When NULL, the prior precision is (sigma_eps^2 / sigma_alpha^2) * I_q matching the diagonal case. Advanced users typically reach re_cov_state via vcmm() rather than calling fit_ss() directly.
x	A vcmm_fit object.
...	Unused; present for S3 method consistency.

Details

At each iteration the algorithm solves:

- beta-update: (C + P) beta = b - XtZ %*% alpha
- alpha-update: (ZtZ + (sigma_eps^2 / sigma_alpha^2) * I) alpha = Zty - t(XtZ) %*% beta

Both linear systems are solved via invert_matrix(), which automatically dispatches between solve() and SVD pseudo-inverse depending on dimension and condition number. Convergence is declared when the relative change in both beta and alpha falls below their respective tolerances.

If control$update_variance is TRUE, the residual variance is re-estimated each iteration from the SS residual sum of squares, and the random-effect variance is re-estimated as mean(alpha^2).

Value

A list of class "vcmm_fit" with elements:

- beta: fitted fixed-effects vector, length p.
- alpha: fitted random-effects vector, length q.
- sigma_eps: final residual standard deviation.
- sigma_alpha: final random-effect standard deviation.
- iterations: number of iterations performed.
- converged: TRUE if convergence tolerances were met.
- elapsed_sec: wall-clock fitting time in seconds.

- `n_obs`, `p`, `q`: data and design dimensions.
- `method`: character, `"SS"`.
- `control`: the `vcmm_control` object used.
- `call`: the matched call.

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

Examples

```
set.seed(1)
n <- 200; p <- 4; q <- 3
X <- cbind(1, matrix(rnorm(n * (p - 1)), n, p - 1))
Z <- matrix(rnorm(n * q), n, q)
alpha_true <- rnorm(q, sd = 0.5)
y <- as.vector(
  X %*% c(2, 0.5, -0.3, 0.8) + Z %*% alpha_true + rnorm(n, sd = 0.5)
)

# Build sufficient statistics and penalty
stats <- compute_sufficient_stats(y, X, Z)
P     <- build_penalty_matrix(n_basis = p - 1, lambda = 0.1)

# Fit with fixed variances
fit <- fit_ss(stats, P,
              vcmm_control(sigma_eps = 0.5, sigma_alpha = 0.5))
fit
coef(fit)
```

`fixef` *Extract fixed-effects from a fitted model object*

Description

Generic in the style of `nlme::fixef` / `lme4::fixef`, redefined here so **cevcmm** avoids a hard dependency on either. If you also have **nlme** or **lme4** loaded, call `cevcmm::fixef(fit)` explicitly.

Usage

```
fixef(object, ...)
```

Arguments

`object`	A model object.
`...`	Method-specific arguments.

Value

The return value depends on the class of `object`; see the appropriate method (e.g. `fixef.vcmm_fit` for `vcmm_fit` objects, which returns a two-element list with scalar `intercept` and a matrix `varying` of B-spline basis coefficients).

`fixef.vcmm_fit` *Fixed effects of a VCMM, reshaped by varying-coefficient block*

Description

Splits the coefficient vector returned by `coef.vcmm_fit` into:

- `intercept`: the constant scalar $\hat{\beta}_0$.
- `varying`: an $m \times K$ matrix of B-spline basis coefficients, with row names `"basis1"`, `...`, `"basisM"` and column names `"X1"`, `...`, `"XK"`.

For $K = 0$ (no varying covariate; intercept-only model), the `varying` slot is `NULL`.

Usage

```
## S3 method for class 'vcmm_fit'
fixef(object, ...)
```

Arguments

`object`	A `vcmm_fit` object.
`...`	Unused.

Value

A two-element list.

See Also

`coef.vcmm_fit`, `varying_coef`, `ranef.vcmm_fit`.

init_accumulator *Initialize an empty sufficient-statistics accumulator*

Description

Allocates a zero-filled accumulator with the correct dimensions to receive batched calls to `accumulate_stats()`. Use this once before a streaming loop over data batches or nodes.

Usage

```
init_accumulator(p, q)
```

Arguments

`p`	Integer. Number of fixed-effects columns (intercept plus spline basis).
`q`	Integer. Number of random-effects columns (length of alpha).

Value

A list of class `"vcmm_accumulator"` with the same six matrix slots as `compute_sufficient_stats()`, all initialised to zero, plus `n_obs = 0L`.

See Also

Other sufficient statistics: `accumulate_stats()`, `compute_sufficient_stats()`

Examples

```
acc <- init_accumulator(p = 5, q = 3)
dim(acc$C)   # 5 x 5
dim(acc$ZtZ) # 3 x 3
acc$n_obs    # 0
```

invert_matrix *Numerically stable matrix inversion with automatic method selection*

Description

Inverts a square matrix A using a dispatch rule that balances speed and numerical stability:

- If `q < 100`: try Cholesky (via `invert_spd_cpp()`), falling through to LU (`invert_general_cpp()`) and finally SVD pseudo-inverse if the matrix is not positive-definite.
- If `q >= 100`: route through the SVD pseudo-inverse path. By default this is the dense LAPACK split-merge variant (`svd_pseudo_inverse()`, paper Algorithm 2). When the user opts in (see the `method` argument below), the iterative truncated SVD from `RSpectra::svds()` is used instead – faster when the matrix has effective rank much smaller than `q`, slower otherwise.

The Cholesky fast path is roughly 2-3x faster than the original `kappa(A) + solve(A)` R path on VCMM K matrices. See the Day 17 and Day 18 validation scripts in `inst/validation/` for the bit-equivalence and timing details.

Usage

```
invert_matrix(
  A,
  q = NULL,
  verbose = FALSE,
  use_cpp = TRUE,
  method = c("auto", "lapack", "rspectra")
)
```

Arguments

`A`	Numeric square matrix to invert.
`q`	Optional integer. Routing dimension used to pick the inversion strategy (defaults to `nrow(A)`). Pass an explicit value if you know `A` is a curvature block with a meaningful dimension that differs from its row count.
`verbose`	Logical. If `TRUE`, print the chosen method.
`use_cpp`	Logical. If `TRUE` (default since Day 17), use the RcppArmadillo Cholesky/LU backend. If `FALSE`, use the original pure-R path with `kappa(A)` plus `solve(A)`.
`method`	Character string, one of `"auto"` (default), `"lapack"`, or `"rspectra"`. Only affects the `q >= 100` branch. With `"auto"`, the routing checks `getOption("cevcmm.use_rspectra", FALSE)`: if `TRUE` and the RSpectra package is installed, the truncated-SVD path is used; otherwise the dense LAPACK split-merge SVD is used. With `"lapack"`, always use the dense LAPACK path (the original Algorithm 2 behaviour). With `"rspectra"`, force the truncated SVD via RSpectra; if the matrix turns out to be full-rank, the function silently falls back to LAPACK so results are always correct.

Details

If you need to reproduce the original R-only path (e.g. for a bit-equivalence test), pass `use_cpp = FALSE`.

Value

A numeric matrix with the same dimensions as `A`.

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

Examples

```
set.seed(1)
A <- crossprod(matrix(rnorm(50), 10, 5)) + diag(5)
A_inv <- invert_matrix(A)
max(abs(A %*% A_inv - diag(5)))  # ~ machine epsilon
```

`logLik.vcmm_fit` *Log-likelihood of a vcmm fit*

Description

Returns the marginal log-likelihood

$$\ell(\hat{\beta}, \hat{\sigma}_\varepsilon, \hat{\Sigma}_\alpha) = -\tfrac{n}{2}\log(2\pi) - \tfrac{1}{2}\log|\Sigma_y| - \tfrac{1}{2}(y - X\hat{\beta})^\top \Sigma_y^{-1}(y - X\hat{\beta}),$$

evaluated at the fitted parameter values, with $\Sigma_y = \sigma_\varepsilon^2 I + Z\,\Sigma_\alpha\,Z^\top$. The value is computed once at convergence and cached on the fit object as `object$marginal_loglik`; this method simply retrieves it and attaches `df` and `nobs` attributes so that `AIC()` and `BIC()` work out of the box.

Usage

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

Arguments

`object`	A `vcmm_fit` object.
`...`	Unused.

Details

Degrees of freedom counted are p (fixed-effects, including all spline basis coefficients) plus the number of free variance-component parameters:

- `re_cov = "diag"`: 2 (σ_ε, σ_α).
- `re_cov = "kronecker"` / `"separable"`: $1 + q_{\text{left}}(q_{\text{left}}+1)/2$ (σ_ε plus the free entries of Σ_{left}). Σ_{right} is held fixed at its user-supplied value and contributes `0` df.

Value

An object of class `"logLik"`; numeric scalar with `df` and `nobs` attributes.

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

Examples

```
set.seed(1)
n <- 400
t <- runif(n); x <- runif(n); Z <- matrix(rnorm(n * 3), n, 3)
y <- 2 + sin(2 * pi * t) * x +
     as.vector(Z %*% rnorm(3, sd = 0.5)) + rnorm(n, sd = 0.5)
fit <- vcmm(y, X = x, Z = Z, t = t,
            control = vcmm_control(sigma_eps = 0.5, sigma_alpha = 0.5))
logLik(fit)
AIC(fit)
BIC(fit)
```

---

nobs.vcmm_fit *Number of observations from a vcmm fit*

Description

Standard `stats::nobs` generic. Returns N, the total number of observations used to compute the fit (or the sum of node sample sizes for fits produced via `fit_from_summaries`).

Usage

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

Arguments

`object`	A `vcmm_fit` object.
`...`	Unused.

Value

Integer.

---

node_summary *Compute one node's sufficient-statistics summary*

Description

Convenience alias for `compute_sufficient_stats`, intended for distributed-computing workflows. Each compute node calls this on its local data and transmits the small returned summary; the central server aggregates summaries with `+` (or `Reduce("+", summaries)`) and fits via `fit_from_summaries`.

Usage

```
node_summary(y, X, Z)
```

Arguments

y	Numeric response vector of length n.
X	Numeric n by p fixed-effects design matrix (intercept plus spline basis columns).
Z	Numeric n by q random-effects design matrix.

Details

What's transmitted. The returned object contains six fixed-size arrays whose dimensions depend only on $p = \mathrm{ncol}(X)$ and $q = \mathrm{ncol}(Z)$, never on the node-local sample size. For a typical VCMM with $p \approx 15$ and $q \approx 50$, one summary is a few kilobytes regardless of N_s.

Basis alignment. All nodes must build their local `X` (the spline-expanded fixed-effects design) using the *same* basis specification. The recommended workflow is to call `build_vcmm_design` once with the full `t`-range (or pre-agreed knots), broadcast the resulting basis, and have each node evaluate it on local `t`. The package itself does not enforce this; mis-aligned bases will silently give incorrect fits.

Value

A `vcmm_ss` object (additive via `+.vcmm_ss`).

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

See Also

`fit_from_summaries`, `compute_sufficient_stats`, `build_vcmm_design`

Examples

```
set.seed(1)
n_per_node <- 100; p <- 5; q <- 3
X <- cbind(1, matrix(rnorm(n_per_node * (p - 1)), n_per_node, p - 1))
Z <- matrix(rnorm(n_per_node * q), n_per_node, q)
y <- rnorm(n_per_node)

gamma_1 <- node_summary(y, X, Z)
gamma_2 <- node_summary(y, X, Z)
gamma_pooled <- gamma_1 + gamma_2
gamma_pooled
```

`plot.vcmm_fit` *Diagnostic plots for a vcmm fit*

Description

Three panels, each requestable via `which`:

1. Varying-coefficient curves $\hat{\beta}_k(t)$ with pointwise Wald confidence bands.
2. Residual diagnostics: residuals vs fitted, residuals vs t. **Requires** `data` since a vcmm fit does not carry the raw training data.
3. Random-effects diagnostics: Normal Q-Q for `re_cov = "diag"`, or a heatmap of the $G \times q_{\text{left}}$ random-effects matrix together with a heatmap of the estimated Σ_{left} for `"kronecker"` / `"separable"`.

Usage

```
## S3 method for class 'vcmm_fit'
plot(
  x,
  which = 1:3,
  data = NULL,
  t_grid = NULL,
  n_grid = 200L,
  conf_level = 0.95,
  ask = (length(which) > 1L) && interactive(),
  ...
)
```

Arguments

`x`	A `vcmm_fit` object.
`which`	Integer vector subset of `1:3`. Default `1:3`.
`data`	Optional list with components `y`, `X`, `Z`, `t` (the training data, or any data on which to compute residuals). Required when panel 2 is requested.
`t_grid`	Numeric. Grid of t values for panel 1. Default `NULL` means an evenly-spaced grid over the stored training range `[t_min, t_max]`.
`n_grid`	Integer. Number of grid points if `t_grid` is `NULL`. Default 200.
`conf_level`	Numeric in (0, 1). Confidence level for panel-1 bands. Default 0.95.
`ask`	Logical. Passed to `devAskNewPage()` when multiple panels are requested in an interactive session.
`...`	Further arguments passed to base graphics calls.

Details

Uses base R graphics, no **ggplot2** dependency. Each requested panel is drawn on its own figure (or set of subfigures via `par(mfrow)`); call `par(mfrow = c(2, 2))` or similar before `plot()` to combine panels in one figure.

Value

Invisibly `NULL`. Called for side effects (plots).

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

See Also

`varying_coef`, `ranef.vcmm_fit`, `predict.vcmm_fit`.

Examples

```
set.seed(1)
n <- 500
t <- runif(n); x <- runif(n); Z <- matrix(rnorm(n * 3), n, 3)
a <- rnorm(3, sd = 0.5)
y <- 2 + sin(2 * pi * t) * x + as.vector(Z %*% a) + rnorm(n, sd = 0.5)
fit <- vcmm(y, X = x, Z = Z, t = t,
            control = vcmm_control(sigma_eps = 0.5, sigma_alpha = 0.5))

plot(fit)                                        # all three panels
plot(fit, which = 1)                             # only varying coefs
plot(fit, which = 2, data = list(y = y, X = x, Z = Z, t = t))
plot(fit, which = 3)                             # ranef diagnostics
```

`predict.vcmm_fit` *Predictions from a fitted VCMM*

Description

Produces predicted responses at new `(X, t)` (and optionally `Z`) values, using the paper's subject-specific predictor

$$\hat{y}_i = \tilde{\mathbf{x}}_i^\top \hat{\tilde{\beta}} + \mathbf{z}_i^\top \hat{\alpha}$$

by default (Jalili and Lin, 2025, Section 5). For the alternative "new groups not seen in training" scenario, pass `include_random = FALSE` to use the marginal predictor $\hat{y}_i = \tilde{\mathbf{x}}_i^\top \hat{\tilde{\beta}}$.

Usage

```
## S3 method for class 'vcmm_fit'
predict(object, newdata, include_random = TRUE, se.fit = FALSE, ...)
```

Arguments

object	A vcmm_fit object.
newdata	A named list or data frame. See Details.
include_random	Logical. If TRUE (default) and newdata\$Z is supplied, adds $Z\hat{\alpha}$ to the prediction. Set FALSE for the marginal predictor (new groups scenario).
se.fit	Logical. If TRUE, also returns per-prediction standard errors.
...	Unused.

Details

newdata format. A named list (or data frame whose columns match these names) containing:

- t: numeric vector of length N_{new}.
- X: numeric matrix $N_{\text{new}} \times K$ (or length-N_{new} vector when $K = 1$) of varying- coefficient covariates, in the same column order used at fit time.
- Z: optional $N_{\text{new}} \times q$ random- effects design matrix. Must follow the same column-stacking convention as the training Z so that Z %*% alpha references the appropriate random-effect slots.

Standard errors. With se.fit = TRUE the per-prediction standard error is the square root of $[W, Z]\,\hat{\sigma}^2_\varepsilon K^{-1}\,[W, Z]^\top$ when include_random = TRUE (joint uncertainty of fixed and random effects), where W is the spline-expanded fixed-effects row; the random-effect block is omitted when include_random = FALSE. These are confidence-interval SEs on the mean; for prediction intervals add $\hat{\sigma}^2_\varepsilon$ to the variance.

Value

Either a numeric vector of length N_{new}, or (when se.fit = TRUE) a list with components fit and se.fit.

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

See Also

varying_coef, vcov.vcmm_fit.

Examples

```
set.seed(1)
n <- 400
t <- runif(n); x <- runif(n); Z <- matrix(rnorm(n * 3), n, 3)
y <- 2 + sin(2 * pi * t) * x +
     as.vector(Z %*% rnorm(3, sd = 0.5)) + rnorm(n, sd = 0.5)

fit <- vcmm(y, X = x, Z = Z, t = t,
            control = vcmm_control(sigma_eps = 0.5, sigma_alpha = 0.5))

# Default: subject-specific predictor (training-group prediction)
yhat_train <- predict(fit, newdata = list(t = t, X = x, Z = Z))
mean((y - yhat_train)^2)  # ~ sigma_eps^2 = 0.25

# Marginal predictor (new groups scenario)
yhat_marg <- predict(fit, newdata = list(t = t, X = x, Z = Z),
                     include_random = FALSE)
```

`ranef` *Extract random effects from a fitted model object*

Description

Generic in the style of `nlme::ranef` / `lme4::ranef`, redefined here so **cevcmm** avoids a hard dependency on either. If you also have **nlme** or **lme4** loaded, call `cevcmm::ranef(fit)` explicitly.

Usage

```
ranef(object, ...)
```

Arguments

`object`	A model object.
`...`	Method-specific arguments.

Value

The return value depends on the class of `object`; see the appropriate method (e.g. `ranef.vcmm_fit` for `vcmm_fit` objects, which returns a numeric vector, matrix, or list reshaped to match the fitted `re_cov` structure).

ranef.vcmm_fit *Random effects of a VCMM, reshaped by re_cov structure*

Description

For `re_cov = "diag"`, returns a named numeric vector of length q. For `"kronecker"` and `"separable"`, reshapes $\hat{\alpha}$ into a $G \times q_{\text{left}}$ matrix (column-stacking convention used throughout the package). Row names are `"g1", ..., "gG"`; column names are `c("origin","dest")` when `re_cov = "kronecker"` and `q_left = 2`, and `"k1", ..., "kK"` otherwise.

Usage

```
## S3 method for class 'vcmm_fit'
ranef(object, ...)
```

Arguments

`object`	A `vcmm_fit` object.
`...`	Unused.

Value

Numeric vector (`"diag"`) or numeric matrix (`"kronecker"` / `"separable"`).

See Also

`coef.vcmm_fit`, `fixef.vcmm_fit`.

summary.vcmm_fit *Summarise a vcmm fit*

Description

Produces a `vcmm_summary` object with a coefficient table for the fixed-effects, the variance components, and basic fit diagnostics. `print()` renders it in the style of `lm` / `lmer`.

Usage

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

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

Arguments

object	A vcmm_fit object.
...	Unused.
x	A vcmm_summary object.
digits	Integer. Number of significant digits to display.
signif.stars	Logical. If TRUE, append significance stars to the coefficient table.

Details

The standard errors are the square roots of the diagonal of vcov(object, which = "beta"), treating σ_ε and σ_α (or Σ_α) as fixed at their fitted values. They are first-order valid (Theorem 3.1) but do not adjust for variance-component uncertainty.

For re_cov %in% c("kronecker", "separable"), the variance- components block of the printed summary displays the estimated Σ_{left} ($\Sigma_{2\times 2}$ for OD or Σ_q for group-shared dense) instead of a scalar σ_α.

Value

A list of class "vcmm_summary".

---

varying_coef *Evaluate the varying coefficient(s) at new t values*

Description

For a VCMM with varying coefficients $\beta_k(t), k = 1, ..., K$, this function evaluates $\hat{\beta}_k(t)$ at any vector of t_new values, using the same B-spline basis the fit was built on. Useful for diagnostic plots, predictions, and reporting; the Day-13 plot method uses it internally.

Usage

```
varying_coef(object, t_new, k = NULL, se.fit = FALSE, ...)
```

Arguments

object	A vcmm_fit object.
t_new	Numeric vector at which to evaluate. Same scale as the original t; the package's normalisation is applied internally.
k	Integer vector of which varying coefficients to evaluate (1-based). Default NULL = all K.
se.fit	Logical. If TRUE, also returns pointwise standard errors.
...	Unused.

Details

The constant intercept $\hat{\beta}_0$ is *not* returned (it does not vary in `t`); use `fixef(fit)\$intercept` for that.

Pointwise standard errors are available via `se.fit = TRUE`, computed as

$$\mathrm{SE}(\hat{\beta}_k(t)) = \sqrt{B(t)^\top \widehat{\mathrm{Var}}(\beta_{(k)})\, B(t)}$$

where $\beta_{(k)}$ is the basis-coefficient sub-vector for coefficient k and the covariance comes from `vcov(object, which = "beta")`.

Value

Either a numeric matrix (`length(t_new)` by `length(k)`, default), or a list with components `fit` and `se.fit` when `se.fit = TRUE`.

See Also

`fixef.vcmm_fit`, `coef.vcmm_fit`.

`vcmm` *Fit a Varying Coefficient Mixed-Effects Model*

Description

The main user-facing fit function. Builds the B-spline design and penalty for one or more varying coefficients in `t`, computes the aggregated sufficient statistics, and fits the model using either the one-step CSL estimator (default) or the iterative SS estimator.

Usage

```
vcmm(
  y,
  X,
  Z,
  t,
  method = c("auto", "csl", "ss"),
  re_cov = c("diag", "kronecker", "separable"),
  n_groups = NULL,
  q_left = NULL,
  Sigma_left_init = NULL,
  Sigma_right_init = NULL,
  Sigma_2x2_init = NULL,
  Sigma_spatial_init = NULL,
  Sigma_q_init = NULL,
  Omega_G_init = NULL,
  n_basis = NULL,
  degree = 3L,
```

```
  lambda = 1,
  control = vcmm_control(),
  normalize_t = TRUE,
  ...
)
```

Arguments

y	Numeric response vector of length n.
X	Numeric $n \times K$ matrix (or length-n vector if $K = 1$) of covariates that get varying coefficients in t.
Z	Numeric $n \times q$ random-effects design matrix. For re_cov = "kronecker" or "separable", q must equal q_left * n_groups.
t	Numeric vector of length n in which the coefficients vary smoothly.
method	Character: "auto" (default), "csl", or "ss".
re_cov	Character: "diag" (default), "kronecker", or "separable". See Details.
n_groups	Integer G. Required for "kronecker" and "separable".
q_left	Integer. The left (within) dimension of the Kronecker factor. For re_cov = "kronecker" defaults to 2 (OD setting); for re_cov = "separable" this is the per-group random-effect dimension and **must be supplied**.
Sigma_left_init	Optional $k \times k$ initial left covariance ($k = q_{\text{left}}$). Aliases accepted: Sigma_2x2_init (for "kronecker" with q_left = 2) and Sigma_q_init (for "separable"). Default is sigma_alpha^2 * I_k.
Sigma_right_init	Optional $G \times G$ initial right covariance. Aliases accepted: Sigma_spatial_init (for "kronecker") and Omega_G_init (for "separable"). Default is I_G.
Sigma_2x2_init	Legacy alias for Sigma_left_init when re_cov = "kronecker" and q_left = 2.
Sigma_spatial_init	Legacy alias for Sigma_right_init when re_cov = "kronecker".
Sigma_q_init	Alias for Sigma_left_init when re_cov = "separable".
Omega_G_init	Alias for Sigma_right_init when re_cov = "separable".
n_basis	Integer or NULL. Number of B-spline basis functions per varying coefficient. NULL (default) auto-picks max(floor(n^(1/3)) + 4, 10).
degree	Integer. B-spline degree (default 3 = cubic).
lambda	Non-negative numeric. Smoothing parameter (default 1).
control	A vcmm_control object.
normalize_t	Logical. If TRUE (default), t is linearly mapped to [0, 1] before building the basis.
...	Further arguments passed to fit_csl() when CSL is used.

Details

Model. For observations $i = 1, \ldots, n$ the fitted model is

$$y_i = \beta_0(t_i) + \sum_{k=1}^{K} x_{ik}\, \beta_k(t_i) + z_i^\top \alpha + \varepsilon_i,$$

where each $\beta_k(t)$ is a cubic B-spline with `n_basis` basis functions and a second-order difference penalty, and $\alpha \sim N(0, \Sigma_\alpha)$ with structure chosen by `re_cov`.

Method selection. The default is `method = "auto"`, which picks `"csl"` when $N \cdot q > 10^5$ or $q > 50$ and `"ss"` otherwise.

Random-effects covariance. Three structures:

- `re_cov = "diag"`: $\alpha \sim N(0, \sigma_\alpha^2 I_q)$.
- `re_cov = "kronecker"`: $\alpha \sim N(0, \Sigma_{\text{left}} \otimes \Sigma_{\text{right}})$ with Σ_{left} of size `q_left` x `q_left` (default `q_left = 2`; OD-style with origin / destination blocks). User-facing names `Sigma_2x2_init`, `Sigma_spatial_init` are accepted as aliases.
- `re_cov = "separable"`: $\alpha \sim N(0, \Sigma_q \otimes \Omega_G)$ with `Sigma_q` of size `q_left` x `q_left` (required, no default) and `Omega_G` of size $G \times G$. User-facing names `Sigma_q_init`, `Omega_G_init` are accepted as aliases for `Sigma_left_init`, `Sigma_right_init`.

Column-stacking convention. For `re_cov = "kronecker"` or `"separable"`, the random-effects vector is $\alpha = \text{vec}_{\text{col}}(M)$ where $M \in \mathbb{R}^{G \times q_{\text{left}}}$, i.e.\ `alpha[(k - 1) * G + g] = M[g, k]`. The `Z` matrix must be constructed so that `Z %*% alpha` gives the correct random-effect contribution. `ncol(Z)` must equal `q_left * n_groups`.

Identifiability of the right component. The right-side covariance (`Sigma_spatial` / `Omega_G`) is not separately identifiable from a single $\hat{\alpha}$, so it is held fixed at the user-supplied initial value (default I_G). Supply a parametric kernel via `Sigma_right_init` (e.g., `exp(-D / phi)` for OD; AR(1) for separable). The left-side covariance (`Sigma_2x2` / `Sigma_q`) is updated every iteration via the EM-style estimator (Theorem 1 M_η rule) when `control$update_variance = TRUE`.

Choosing `re_cov`. The three modes specify the assumed covariance structure of the random-effects vector α; they do **not** constrain what `Z`'s entries look like. The distribution of `Z` (binary indicators, continuous values, mixed) does not enter this choice — only the structure of α does. Pick by data shape:

- `"diag"`: independent random effects, one per group or subject, no assumed cross-group dependence. Simplest case. Use when each group contributes a single offset and groups are exchangeable.
- `"kronecker"`: origin-destination flow data. Every row of `Z` activates one origin indicator and one destination indicator, so `ncol(Z) = 2 * G` where G is the number of regions. The 2x2 left block captures origin/destination dependence; the $G \times G$ right block captures spatial dependence between regions.
- `"separable"`: each group carries multiple correlated random effects (`q_left > 2`) — for example a random intercept plus a random slope per region. Use when the per-group random-effect dimension exceeds 2.

Value

A vcmm_fit object as returned by fit_ss() or fit_csl(), augmented with design (basis metadata) and re_cov. When re_cov is "kronecker" or "separable", re_cov_state contains the canonical fields Sigma_left, Sigma_right, plus legacy aliases Sigma_2x2/Sigma_spatial (when q_left = 2) or Sigma_q/Omega_G (for separable).

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

Examples

```
set.seed(1)
n <- 500
t  <- runif(n)
x  <- runif(n)
Z  <- matrix(rnorm(n * 3), n, 3)
alpha_true <- rnorm(3, sd = 0.5)
y  <- 2 + sin(2 * pi * t) * x +
      as.vector(Z %*% alpha_true) + rnorm(n, sd = 0.5)

fit <- vcmm(y, X = x, Z = Z, t = t,
            control = vcmm_control(sigma_eps = 0.5, sigma_alpha = 0.5))
fit
```

---

vcmm_control *Control parameters for VCMM fitting*

Description

Builds a validated options list controlling the iterative SS estimator (and, later, the CSL and SVD-stabilised estimators). Pass the returned object as the control argument of fit_ss().

Usage

```
vcmm_control(
  max_iter = 200L,
  tol_beta = 1e-06,
  tol_alpha = 1e-06,
  sigma_eps = 1,
  sigma_alpha = 1,
  update_variance = FALSE,
  verbose = FALSE
)

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

Arguments

max_iter	Integer. Maximum number of iterations (default 200).
tol_beta	Positive numeric. Relative-change convergence tolerance for the fixed-effects coefficient vector beta (default 1e-6).
tol_alpha	Positive numeric. Relative-change convergence tolerance for the random-effects vector alpha (default 1e-6).
sigma_eps	Positive numeric. Initial residual standard deviation. If update_variance = FALSE, this value is held fixed throughout fitting (default 1).
sigma_alpha	Positive numeric. Initial random-effect standard deviation. If update_variance = FALSE, this value is held fixed throughout fitting (default 1).
update_variance	Logical. If FALSE (default), sigma_eps and sigma_alpha are held fixed at the supplied values – matching Algorithm 1 of Jalili and Lin (2025) as written. If TRUE, both are re-estimated at every iteration using the residual sum of squares formula (for sigma_eps) and a method-of-moments update (for sigma_alpha).
verbose	Logical. If TRUE, print progress every 20 iterations (default FALSE).
x	A vcmm_control object.
...	Unused.

Value

A list of class "vcmm_control" containing the validated options.

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

Examples

```
# Defaults: fix variances at 1, iterate up to 200 times.
ctrl <- vcmm_control()
ctrl

# Fix variances at user-supplied values.
ctrl <- vcmm_control(sigma_eps = 0.5, sigma_alpha = 0.5)

# Re-estimate variances each iteration.
ctrl <- vcmm_control(sigma_eps = 0.5, sigma_alpha = 0.5,
                     update_variance = TRUE)
```

`vcov.vcmm_fit` *Variance-covariance matrix of the fixed-effects from a vcmm fit*

Description

Returns the asymptotic variance-covariance matrix of the fixed-effects coefficient vector $\hat{\beta}$ from a fitted `vcmm_fit`. The matrix is computed as

$$\widehat{\mathrm{Var}}(\hat{\beta}) = \hat{\sigma}^2_{\varepsilon} \cdot [K^{-1}]_{1:p,\,1:p},$$

where K is the prior-augmented Hessian assembled at convergence and cached in `object$K_inv`. This is the standard plug-in asymptotic-normal variance estimator for the linear normal VCMM with fixed variance components.

Usage

```
## S3 method for class 'vcmm_fit'
vcov(object, which = c("beta", "alpha", "both"), ...)
```

Arguments

`object`	A `vcmm_fit` object.
`which`	Character: `"beta"` (default), `"alpha"`, or `"both"`.
`...`	Unused.

Details

Pass `which = "alpha"` for the random-effect block, `which = "both"` for the full $(p+q) \times (p+q)$ joint matrix.

Value

A numeric matrix:

- `"beta"`: p by p.
- `"alpha"`: q by q.
- `"both"`: (p+q) by (p+q), joint.

References

Jalili, L. and Lin, L.-H. (2025). Scalable and Communication-Efficient Varying Coefficient Mixed-Effects Models.

Examples

```
set.seed(1)
n <- 300
t <- runif(n); x <- runif(n)
Z <- matrix(rnorm(n * 3), n, 3)
y <- 2 + sin(2 * pi * t) * x +
     as.vector(Z %*% rnorm(3, sd = 0.5)) + rnorm(n, sd = 0.5)

fit <- vcmm(y, X = x, Z = Z, t = t,
            control = vcmm_control(sigma_eps = 0.5, sigma_alpha = 0.5))

V_beta <- vcov(fit)
dim(V_beta)
```

Index