Package 'stLMM'

July 28, 2026

Type Package

Title Bayesian Spatial and Space-Time Linear Mixed Models

Version 0.0.2

Author Andrew O. Finley [aut, cre],
Sudipto Banerjee [ctb],
Abhirup Datta [ctb],
Paul B. May [ctb]

Maintainer Andrew O. Finley <finleya@msu.edu>

Description Fits Bayesian linear mixed models for spatial and space-time data with fixed effects, independent and identically distributed (iid) grouped random effects, and structured latent processes. The formula interface supports first-order autoregressive (AR(1)) effects, dense Gaussian processes, nearest-neighbor Gaussian processes, proper and Leroux conditional autoregressive (CAR) effects, ordered directed acyclic graph autoregressive (DAGAR) effects, separable CAR-time and DAGAR-time effects, and spatially varying coefficients. The sampler uses sparse precision matrix calculations when available and includes post-fitting tools for latent process recovery, fitted values, prediction, pointwise log likelihoods, and posterior sample extraction. Method details include Datta et al. (2016) <doi:10.1080/01621459.2015.1044091>, Finley et al. (2019) <doi:10.1080/10618600.2018.1537924>, Datta et al. (2019) <doi:10.1214/19-BA1177>, and May and Finley (2025) <doi:10.1016/j.spasta.2025.100917>.

URL https://github.com/finleya/stLMM, https://finleya.github.io/stLMM/

BugReports https://github.com/finleya/stLMM/issues

License GPL (>= 3)

Encoding UTF-8

LazyData true

Depends R (>= 4.4.0)

Imports BayesLogit (>= 2.4), coda, Matrix (>= 1.7-0), stats

LinkingTo BayesLogit (>= 2.4), Matrix (>= 1.7-0)

Suggests ggplot2, knitr, loo, quarto, sf, testthat (>= 3.0.0)

VignetteBuilder quarto

Config/testthat/edition 3

Config/Needs/website dplyr, lme4, nimble, patchwork, spdep, tidyr

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-07-28 16:30:03 UTC

Contents

`as_mcmc` *Convert stLMM samples to coda objects*

Description

Convert retained posterior samples to **coda** objects for chain diagnostics and plotting.

Usage

```
as_mcmc(object, ...)
```

Arguments

`object`	An `stLMM`, `stLMM_chains`, `stLMM_recovery`, or `stLMM_recovery_chains` object.
`...`	Additional arguments passed to methods. Supported arguments include `include_w`, logical, `burn`, a nonnegative integer, and `thin`, a positive integer.

Details

Single-chain fits are returned as `coda::mcmc`. Multi-chain fits are returned as `coda::mcmc.list`. By default, saved or recovered latent process samples `w` are omitted because they can be high-dimensional and are often not needed for covariance-parameter diagnostics.

For ordinary fitted objects, `burn` and `thin` select retained MCMC rows by row number. For recovered objects, `include_w = TRUE` aligns parameter samples to `recover_iter` before appending saved or recovered latent process draws. In that case, `burn` is interpreted on the original posterior iteration scale and `thin` is applied to the recovered draws that remain after burn-in. Multi-chain recovery objects apply the same rule within each chain and return a `coda::mcmc.list`.

Value

A `coda::mcmc` or `coda::mcmc.list` object containing active posterior sample blocks such as fixed effects, grouped random effects, residual variance parameters, process variances, and process correlation parameters.

See Also

`as_samples`, `recover`, `stLMM`, `summary.stLMM`

Examples

```
set.seed(1)
dat <- data.frame(y = rnorm(8), x = seq(-1, 1, length.out = 8))
fit <- stLMM(y ~ x, data = dat, n_samples = 8, warmup = FALSE, verbose = FALSE)
m <- as_mcmc(fit, burn = 2)
coda::effectiveSize(m)
```

---

as_samples *Collect stLMM posterior samples in a data frame*

Description

Collect posterior sample matrices from fitted, recovered, prediction, or fitted-value objects into a plain data frame with samples in rows and named sample columns across the top.

Usage

```
as_samples(object, ...)
```

Arguments

`object`	An `stLMM`, `stLMM_chains`, `stLMM_recovery`, `stLMM_recovery_chains`, `stLMM_prediction`, `stLMM_prediction_chains`, `stLMM_fitted_chains`, or posterior sample matrix.
`...`	Additional arguments passed to methods. Supported arguments include `burn`, `thin`, `metadata`, `include_w`, `combine_chains`, and, for prediction objects, `sample`.

Details

For fitted stLMM objects, columns contain active parameter sample blocks such as fixed effects, grouped random effects, residual variance parameters, process variances, and process correlation parameters. Structured latent process draws are available when they were saved during fitting or added by recover(), and are included only when include_w = TRUE.

For recovery objects, parameter samples are aligned to object$recover_iter before saved or recovered latent process draws are appended. This ensures parameter columns and w_ columns describe the same retained posterior iterations.

For multi-chain objects, samples are combined by row by default and the .chain metadata column preserves chain membership. Set combine_chains = FALSE to return one data frame per chain.

Prediction methods accept sample = "mu", "y", or "all". Prediction columns are named mu_1, mu_2, and so on, or y_1, y_2, and so on.

Value

A data frame, or a list of data frames when combine_chains = FALSE. When metadata = TRUE, the first columns are .chain and .iteration. .iteration is the original MCMC iteration within chain when that metadata is available.

See Also

as_mcmc, recover, predict.stLMM_recovery, fitted

Examples

```
set.seed(1)
dat <- data.frame(y = rnorm(8), x = seq(-1, 1, length.out = 8))
fit <- stLMM(y ~ x, data = dat, n_samples = 8, warmup = FALSE, verbose = FALSE)
draws <- as_samples(fit, burn = 2, thin = 2)

newdat <- data.frame(x = c(-0.5, 0.5))
pred <- predict(fit, newdata = newdat, y_samples = TRUE)
pred_draws <- as_samples(pred, sample = "all")
```

---

car_graph *Build a CAR adjacency graph*

Description

Normalizes an areal adjacency structure for use in car(), dagar(), car_time(), and dagar_time() terms in stLMM formulas.

Usage

```
car_graph(graph, id = NULL, queen = TRUE, island = c("error", "nearest"),
  island_k = 1L)

## S3 method for class 'stLMM_car_graph'
plot(x, show_ids = FALSE, show_nodes = TRUE,
  show_bridges = TRUE, color_by_degree = FALSE, ...)

## S3 method for class 'stLMM_graph'
plot(x, show_ids = FALSE, show_nodes = TRUE,
  color_by_degree = FALSE, ...)
```

Arguments

graph	An sf polygon object or a square symmetric adjacency matrix.
x	A graph object returned by car_graph() or an internal stLMM_graph object stored in a fitted model backend.
id	For sf inputs, the polygon ID column used to match car(area_id, graph = ...), dagar(area_id, graph = ...), or dagar_time(area_id, time, graph = ...) values.
queen	For sf inputs, whether to use queen-style boundary touching. If FALSE, rook-style edge sharing is used.
island	Policy for disconnected polygon components. The default "error" reports disconnected or isolated areas. "nearest" adds nearest-neighbor bridge edges between disconnected sf polygon components.
island_k	Positive integer number of nearest bridge edges to add from the selected disconnected component at each connection step when island = "nearest".
show_ids	Logical; if TRUE, draw graph IDs at node locations.
show_nodes	Logical; if TRUE, draw representative node points.
show_bridges	Logical; if TRUE, bridge edges added by island = "nearest" are highlighted.
color_by_degree	Logical; if TRUE, color polygons or nodes by graph degree.
...	Additional arguments passed to the internal graph plotting helper.

Details

The graph stores binary adjacency, not a row-standardized matrix. The CAR implementation uses a degree-scaled proper CAR precision proportional to D - rho W, where D is the diagonal degree matrix and W is the symmetric binary adjacency matrix. This is equivalent to a CAR conditional specification with row-standardized neighbor averaging in the conditional mean, $E(w_i \mid w_{-i}) = rhod_i^{-1} \sum_{j \sim i} w_j$, with conditional variance $sigma_sq/d_i$. The process variance sigma_sq is supplied as the model term's variance parameter. Isolated areas currently error. Disconnected components also error by default.

For matrix input, row and column names define the graph IDs. If names are not present, integer IDs are used. Matrix input must be square, symmetric, and have no isolated rows. Nonzero off-diagonal

entries are treated as adjacency indicators and are stored as binary edges. Nearest-neighbor island bridging is not available for matrix input because no polygon geometry is available.

For `sf` input, `id` names the polygon identifier column. The helper uses polygon boundary relationships to build a binary adjacency matrix. With `queen = TRUE`, polygons that touch at a point or along an edge are neighbors. With `queen = FALSE`, polygons must share an edge.

With `island = "nearest"`, disconnected components are connected by artificial nearest-neighbor bridge edges based on polygon representative-point distances. These edges are stored in `island_added_edges` so the graph construction choice is auditable.

For teaching and auditability, a useful workflow is to first call `car_graph()` with the default `island = "error"` and then rerun with `island = "nearest"` only when disconnected island components are expected.

The `plot()` methods return `ggplot2` objects. For `sf`-based graphs, polygons are drawn with graph edges overlaid. For matrix-based graphs, a deterministic circular node layout is used. Internal DAGAR graph objects are drawn with directed arrows from each parent node to its child node, making the ordering-induced directed graph visible.

Value

An object of class `stLMM_car_graph`. The object stores graph IDs, degree information, and lower-triangle edge pairs used by `car()`, `dagar()`, `car_time()`, and `dagar_time()`. Island-bridging metadata are stored in `island_policy`, `island_k`, `island_components_initial`, and `island_added_edges`.

See Also

`stunitco`, `stLMM`, `stLMM-terms`

Examples

```
adj <- matrix(
  c(
    0, 1, 0, 0,
    1, 0, 1, 0,
    0, 1, 0, 1,
    0, 0, 1, 0
  ),
  nrow = 4,
  byrow = TRUE,
  dimnames = list(paste0("a", 1:4), paste0("a", 1:4))
)
g <- car_graph(adj)
```

`fitted.stLMM` *Posterior fitted values for an stLMM fit*

Description

Computes posterior fitted values for the original data used to fit an stLMM model. These fitted values include all model components that were fit: formula offsets, fixed effects, explicit grouped random effects, and saved or recovered structured latent process effects. Binomial fitted values default to posterior fitted probabilities, and negative-binomial fitted values default to posterior fitted mean counts.

Usage

```
## S3 method for class 'stLMM'
fitted(object, summary = TRUE,
  sub_sample = list(start = 1L, thin = 1L), scale = c("response", "link"),
  ...)
```

Arguments

object	An object returned by stLMM or recover.
summary	Logical; if TRUE, return posterior mean fitted values. If FALSE, return the posterior fitted-value sample matrix.
sub_sample	List with optional start and thin entries used to subset posterior draws. For models with structured process terms, this subsets the saved or recovered latent-process draw indices in object$recover_iter.
scale	For family = "binomial", return fitted probabilities on the "response" scale or the linear predictor on the "link" scale. For family = "negative_binomial", return fitted mean counts on the "response" scale or the log mean on the "link" scale. For Gaussian models these scales are identical.
...	Currently unused.

Details

This method does not predict at new locations. For models with structured process terms, fitted values require saved or recovered latent process samples. For Polya-Gamma process models, stLMM() saves in-chain process draws by default as save_process = list(start = 1, thin = 1) and recover() subsets them. For Gaussian process models, call recover() on the fitted object to make those samples available.

Value

If summary = TRUE, a numeric vector of posterior mean fitted values for the original fitted data rows, including rows whose response was missing at fit time. If summary = FALSE, a matrix with posterior draws in rows and original data rows in columns. The matrix has a draw_index attribute giving the original MCMC iteration indices used and, for summary = FALSE Polya-Gamma fits, a scale attribute.

get_cor_models *List supported covariance models*

Description

Returns metadata for covariance models supported by the current sampler. This is a lightweight discovery helper for valid `cov_model` values and the theta parameters expected by each model.

Usage

```
get_cor_models()
```

Details

The `domains` entry is used to validate theta priors. Positive theta parameters require finite positive support. Unit theta parameters have support within $[0, 1]$.

The returned metadata applies to covariance-model labels used by `gp()` and `nngp()` terms. CAR-family terms have fixed parameter names documented in `stLMM_terms`.

Value

A named list describing available covariance models. Each entry contains the model's parameter names, parameter type codes, theta domains, and distance mode code.

Examples

```
names(get_cor_models())
get_cor_models()[["exp"]]
get_cor_models()[["matern"]]
```

log_lik *Pointwise log likelihood and WAIC*

Description

`log_lik()` returns the pointwise log-likelihood matrix used for model-fit diagnostics. `waic()` computes Watanabe-Akaike information criterion using `loo::waic()`.

Usage

```
log_lik(object, ...)

## S3 method for class 'stLMM'
log_lik(object,
  sub_sample = list(start = 1L, thin = 1L), ...)

## S3 method for class 'stLMM_chains'
log_lik(object,
  sub_sample = list(start = 1L, thin = 1L), ...)

## S3 method for class 'stLMM_recovery'
log_lik(object,
  sub_sample = list(start = 1L, thin = 1L), ...)

## S3 method for class 'stLMM_recovery_chains'
log_lik(object,
  sub_sample = list(start = 1L, thin = 1L), ...)

waic(object, ...)

## S3 method for class 'stLMM'
waic(object,
  sub_sample = list(start = 1L, thin = 1L), ...)

## S3 method for class 'stLMM_chains'
waic(object,
  sub_sample = list(start = 1L, thin = 1L), ...)

## S3 method for class 'stLMM_recovery'
waic(object,
  sub_sample = list(start = 1L, thin = 1L), ...)

## S3 method for class 'stLMM_recovery_chains'
waic(object,
  sub_sample = list(start = 1L, thin = 1L), ...)
```

Arguments

`object`	An `stLMM` fit, recovered `stLMM_recovery` object, or corresponding multi-chain object. Models with structured process terms require saved or recovered latent process samples.
`sub_sample`	A list with optional integer entries `start` and `thin`. For ordinary fits these select posterior draw indices. For recovered objects and Polya-Gamma process fits with saved process draws, these select saved or recovered draws through `recover_iter`.
`...`	Additional arguments. For `waic()`, these are passed to `loo::waic()`.

Details

The returned log-likelihood matrix has posterior draws in rows and observed response rows in columns. Missing response rows are excluded because they do not contribute to the fitted likelihood.

The calculation is conditional on the latent model expression. For Gaussian models, each entry is evaluated with the fitted linear predictor and the draw-specific residual standard deviation. For binomial models, entries are computed with the inverse-logit probability and fitted trial count. For fixed-size negative-binomial models, entries are computed with the log-mean linear predictor and the fitted size parameter.

For models with structured process terms, `log_lik()` requires saved or recovered latent process samples so that the latent linear predictor can be evaluated draw by draw. Gaussian process models should therefore be passed to `recover()` before calling `log_lik()` or `waic()`. Polya-Gamma process fits can be used directly when process draws were saved during `stLMM()`.

`waic()` requires the optional **loo** package. **loo** is suggested, not imported, so ordinary model fitting does not require it.

Value

`log_lik()` returns a numeric matrix. `waic()` returns the object produced by `loo::waic()`.

See Also

`stLMM`, `recover`, `fitted`, `predict.stLMM_recovery`

Examples

```
set.seed(1)
dat <- data.frame(y = rnorm(8), x = seq(-1, 1, length.out = 8))
fit <- stLMM(y ~ x, data = dat, n_samples = 8, warmup = FALSE, verbose = FALSE)
ll <- log_lik(fit, sub_sample = list(start = 5, thin = 1))

if (requireNamespace("loo", quietly = TRUE)) {
  waic(fit, sub_sample = list(start = 5, thin = 1))
}
```

---

predict.stLMM_recovery

Posterior prediction from recovered stLMM fits

Description

Returns posterior samples for the mean response at fitted rows or at `newdata` rows. For binomial models, the default mean response is the posterior fitted probability. For negative-binomial models, the default mean response is the posterior fitted mean count.

Usage

```
## S3 method for class 'stLMM'
predict(object, newdata = NULL, y_samples = FALSE,
  n_omp_threads = NULL, verbose = FALSE, sub_sample = list(start = 1L,
  thin = 1L), scale = c("response", "link"), ...)

## S3 method for class 'stLMM_chains'
predict(object, newdata = NULL, y_samples = FALSE,
  n_omp_threads = NULL, verbose = FALSE, sub_sample = list(start = 1L,
  thin = 1L), scale = c("response", "link"), ...)

## S3 method for class 'stLMM_recovery'
predict(object, newdata = NULL, y_samples = FALSE,
  joint = FALSE, joint_method = c("full", "vecchia"), pred_m = NULL,
  pred_ordering = "maxmin", st_scale = NULL,
  return_w_samples = TRUE, n_omp_threads = NULL, verbose = FALSE,
  sub_sample = list(start = 1L, thin = 1L), scale = c("response", "link"),
  ...)

## S3 method for class 'stLMM_recovery_chains'
predict(object, newdata = NULL,
  y_samples = FALSE, joint = FALSE, joint_method = c("full", "vecchia"),
  pred_m = NULL, pred_ordering = "maxmin", st_scale = NULL,
  return_w_samples = TRUE, n_omp_threads = NULL, verbose = FALSE,
  sub_sample = list(start = 1L, thin = 1L), scale = c("response", "link"),
  ...)
```

Arguments

`object`	An `stLMM` fit without structured process terms, a Polya-Gamma process fit with saved in-chain process draws, a recovered object returned by `recover()`, or the corresponding multi-chain object.
`newdata`	Optional data frame. Grouped random-effect levels must already exist in the fitted model support. For recovered process models, process coordinates may be existing support locations or new AR1, GP, or NNGP nodes. CAR and CAR-time prediction rows must use existing graph areas; CAR-time rows must also use fitted time values.
`y_samples`	Logical; if `TRUE`, also simulate observation-level posterior predictive samples. Gaussian fits use the fitted residual variance. Binomial fits draw counts from the fitted probabilities and prediction trial counts.
`joint`	Logical; for new GP or NNGP nodes, whether to draw unique new nodes with cross-location dependence. The default `FALSE` simulates unique new nodes independently conditional on the fitted support or fitted-support neighbors and is recommended when only marginal prediction summaries are needed. For dense `gp()` terms, `joint = TRUE` uses the exact dense GP conditional covariance among unique new nodes.

`joint_method` Method used for NNGP terms when `joint = TRUE`. `"full"` keeps the historical behavior: fitted-support neighbors are used for each new node and the new-node residual covariance is drawn from a dense matrix. `"vecchia"` orders the unique new prediction nodes and simulates them sequentially from a Vecchia/NNGP-style history containing all fitted support nodes plus previously simulated prediction nodes. Dense `gp()` terms ignore `joint_method` and use exact joint Gaussian conditioning when `joint = TRUE`.

`pred_m` Optional positive integer neighbor count for `joint_method = "vecchia"`. The default `NULL` uses the fitted NNGP term's `m`. The first prediction nodes always have the fitted support available; if fewer than `pred_m` history nodes are available, all available history nodes are used.

`pred_ordering` Ordering for unique new prediction nodes under `joint_method = "vecchia"`. May be one of the NNGP ordering options `"coord"`, `"default"`, `"maxmin"`, `"hilbert"`, `"random"`, or a numeric permutation of the unique new prediction nodes. For space-time NNGP terms, `"hilbert"` is currently rejected because the implemented Hilbert ordering is two-dimensional.

`st_scale` `NULL`, a positive scalar, or a named positive vector/list by process term. For space-time NNGP new-node prediction, `NULL` inherits the `st_scale` stored on the fitted NNGP graph, falling back to `1` for older objects. A supplied value overrides the fitted scale. The final coordinate is multiplied by `st_scale` before prediction ordering and nearest-neighbor ranking. This is not automatic standardization: spatial coordinates are left on their original scale and the time coordinate is left on its original scale except for this single multiplier during graph construction. The covariance calculation still uses the original coordinates. Non-default `st_scale` values are valid only for space-time NNGP terms.

`return_w_samples` Logical; for recovered process prediction, whether to return raw latent process samples for each process term in `w_samples`. The default `TRUE` is useful for diagnostics and process maps. Set to `FALSE` for large prediction problems when memory use matters. For covariate-scaled process terms, returned values are the raw latent process draws, not the covariate-multiplied row-level contribution.

`n_omp_threads` Optional positive integer number of OpenMP threads for threaded prediction kernels. The default `NULL` uses the thread count stored in the fitted or recovered object. This affects threaded NNGP prediction neighbor searches and compiled NNGP prediction simulation kernels; R-level dense joint GP and dense joint NNGP prediction are not OpenMP-threaded.

`verbose` Logical; if `TRUE`, print phase-level prediction progress messages. Progress is reported from R before and after major prediction phases rather than from individual OpenMP worker threads.

`sub_sample` A list with optional integer entries `start` and `thin`. For `predict.stLMM()` without process terms, these select posterior draw indices. For Polya-Gamma process fits with saved process draws, and for `predict.stLMM_recovery()`, these select saved or recovered draw indices through `object$recover_iter`.

`scale` For `family = "binomial"`, return fitted probabilities on the `"response"` scale or the linear predictor on the `"link"` scale. For `family = "negative_binomial"`, return fitted mean counts on the `"response"` scale or the log mean on the `"link"` scale. For Gaussian models these scales are identical.

... Currently unused.

Details

Prediction always represents the model as fit. Prediction-time dropping of model components is not supported. New grouped random-effect levels are not supported.

By default, prediction uses the OpenMP thread count stored in the fitted or recovered object. Use `n_omp_threads` to override that value for a prediction call without refitting or re-running `recover()`. Because the long-running prediction kernels can be OpenMP threaded, `verbose = TRUE` reports progress at the R phase level rather than printing from worker threads.

With `newdata = NULL`, prediction returns fitted-row posterior mean samples for the original data rows. With `newdata`, fixed-effect and iid random-effect design matrices are rebuilt from the fitted formula metadata. If the fitted formula contains `offset()`, the same offset expression is evaluated in `newdata` and added to the prediction linear predictor. New AR1, GP, and NNGP process nodes are supported. Repeated prediction rows sharing a new process node reuse the same simulated latent draw within a posterior sample.

When `return_w_samples = TRUE`, recovered-process prediction objects also contain `w_samples`, a named list with one matrix per process term. Each matrix has posterior draws in rows and prediction rows in columns, aligned with `mu_samples`. These are raw latent process values on the prediction rows. For spatially varying coefficient terms, `w_samples` does not include the covariate scaling used when adding the process contribution to `mu_samples`.

For `family = "binomial"`, prediction first constructs posterior samples of the linear predictor and then applies the inverse-logit transform when `scale = "response"`. If `y_samples = TRUE`, prediction simulates binomial counts. With `newdata = NULL`, the fitted trial counts are used. With `newdata`, a column named `trials` is used when present; otherwise prediction defaults to one trial per row.

For `family = "negative_binomial"`, prediction first constructs posterior samples of the log mean and then exponentiates them when `scale = "response"`. If `y_samples = TRUE`, prediction simulates negative-binomial counts using the fitted fixed `size`.

For dense `gp()` terms, `joint = FALSE` draws each unique new node from its marginal conditional distribution given the fitted support. With `joint = TRUE`, unique new GP nodes are drawn jointly from the exact conditional Gaussian distribution using the dense covariance among prediction nodes and fitted nodes. This is appropriate for small prediction sets and is dense in the number of unique new GP nodes.

For NNGP terms, `joint = FALSE` uses the independent new-node prediction approach based on nearest fitted-support neighbors, following the conditional NNGP prediction machinery described by Finley et al. (2019). Each unique new node is simulated independently conditional on its fitted-support neighbors. With `joint = TRUE, joint_method = "full"`, nearest neighbors are still chosen only from the fitted support, but unique new NNGP nodes are drawn jointly from a dense residual covariance. This can require substantial memory for large prediction sets.

With `joint = TRUE, joint_method = "vecchia"`, unique new NNGP nodes are ordered by `pred_ordering`. Neighbors for each ordered prediction node are chosen from the combined history made of all fitted support nodes and earlier prediction nodes in that ordering. The prediction draw is then generated sequentially, so simulated earlier prediction nodes can enter the conditional mean of later prediction nodes. This avoids a dense new-node covariance matrix and is usually the more scalable joint

prediction method. This option follows the observed-prediction ordering idea and latent-first full-conditioning construction described by Katzfuss et al. (2020, Sects. 3.1, 4.2, and 4.3.4), adapted to the package workflow in which fitted latent process draws are saved or recovered first and prediction nodes are then simulated sequentially conditional on those draws. The method is ordering dependent, as in other Vecchia/NNGP approximations.

The independent, dense-joint, and Vecchia-joint approaches should have similar posterior means and medians for each prediction location when the neighbor sets are adequate. They differ mainly in the cross-location dependence of the simulated prediction draws.

For space-time NNGP prediction, `st_scale` controls the search geometry used to order new nodes and choose nearest neighbors. The search distance is equivalent to Euclidean distance after replacing the final coordinate by `st_scale * time`. Thus `st_scale = 1` uses the coordinates as supplied, while `st_scale = s` treats one unit of the final coordinate as `s` spatial-coordinate units for graph construction. Larger values make temporal separation matter more and tend to choose neighbors closer in time; smaller values make temporal separation matter less. This choice does not change the covariance parameters or covariance evaluations once the neighbors have been selected.

Value

An object of class `stLMM_prediction`, a list with `mu_samples`, optional `y_samples`, optional `w_samples`, `draw_index`, and metadata. Binomial prediction objects also include `scale`. For multi-chain input, the returned object has class `stLMM_prediction_chains` and contains `chains`, a list of ordinary `stLMM_prediction` objects. The `sub_sample` rule is applied within each chain.

References

Finley, A. O., Datta, A., Cook, B. D., Morton, D. C., Andersen, H. E., and Banerjee, S. (2019). Efficient Algorithms for Bayesian Nearest Neighbor Gaussian Processes. *Journal of Computational and Graphical Statistics*, 28(2), 401–414. doi:10.1080/10618600.2018.1537924.

Katzfuss, M., Guinness, J., Gong, W., and Zilber, D. (2020). Vecchia Approximations of Gaussian-Process Predictions. *Journal of Agricultural, Biological and Environmental Statistics*, 25(3), 383–414. doi:10.1007/s13253020004017.

See Also

`recover`, `fitted`, `summary.stLMM_prediction`

Examples

```
set.seed(1)
dat <- data.frame(
  y = rnorm(8),
  x = rnorm(8),
  time = seq_len(8)
)

fit <- stLMM(
  y ~ x + ar1(time),
  data = dat,
  n_samples = 8,
```

```
  priors = list(
    resid = list(tau_sq = ig(2, 1)),
    ar1_1 = list(sigma_sq = ig(2, 1), phi = uniform(-0.8, 0.8))
  ),
  warmup = FALSE,
  verbose = FALSE
)
rec <- recover(fit)
pred_fit <- predict(rec)
newdat <- data.frame(x = c(-0.5, 0.5), time = c(9, 10))
pred_new <- predict(rec, newdata = newdat, y_samples = TRUE)
summary(pred_new)
```

`recover` *Recover latent process draws*

Description

Recover posterior draws of latent process terms from a fitted collapsed `stLMM` model.

Usage

```
recover(object, ...)

## S3 method for class 'stLMM'
recover(object, n_omp_threads = NULL, verbose = FALSE,
  sub_sample = list(start = 1L, thin = 1L), ...)

## S3 method for class 'stLMM_chains'
recover(object, n_omp_threads = NULL, verbose = FALSE,
  sub_sample = list(start = 1L, thin = 1L), ...)
```

Arguments

`object`	An object returned by `stLMM()`. Multi-chain fits are handled chain by chain.
`n_omp_threads`	Optional positive integer number of OpenMP threads for threaded recovery setup kernels. The default `NULL` uses the thread count stored in the fitted object. This mainly affects NNGP covariance-coefficient updates needed while reconstructing latent process draws; sparse CHOLMOD factorization and random-number generation are not OpenMP-threaded by `recover()`.
`verbose`	Logical; if `TRUE`, print phase-level recovery progress messages. Progress is reported from R before and after major recovery phases rather than from individual OpenMP worker threads.
`sub_sample`	A list with optional integer entries `start` and `thin` selecting posterior draws to recover or select from saved process draws.
`...`	Currently unused.

Details

For Gaussian process models, the collapsed sampler integrates latent process values out during fitting. This function reconstructs the posterior conditional distribution of the latent process for selected posterior draws and samples from it using the stored backend, posterior parameter samples, and CHOLMOD sparse factorization.

By default, recovery uses the OpenMP thread count stored in the fitted object. Use `n_omp_threads` to override that value for a recovery call without refitting. The override is most relevant for NNGP terms because their covariance-coefficient updates are threaded. Other major recovery work, including sparse factorization and latent Gaussian random draws, is not parallelized by this argument.

For Polya-Gamma process models, process draws are saved during `stLMM()` by default as `save_process = list(start = 1, thin = 1)`. In that case, `recover()` only selects and labels the saved in-chain process draws. If a Polya-Gamma process model was fit with `save_process = FALSE`, `recover()` errors and the model must be refit with `save_process = TRUE` or `save_process = list(start = ..., thin = ...)`.

Recovery is only needed for models with structured process terms such as `ar1()`, `gp()`, `nngp()`, `car()`, `dagar()`, or `car_time()`, or `dagar_time()`. Explicit iid grouped random effects are sampled during fitting and do not need this recovery step. When the fitted data include rows with missing numeric responses, recovered process samples are on the full latent support, including latent nodes associated only with those missing-response rows.

For Polya-Gamma process models, the sampler uses warm-running internal Polya-Gamma latent-process draws to update the augmentation variables. Those same in-chain process draws are the process samples used by `recover()`. Post-fit Polya-Gamma reconstruction is intentionally not supported.

Value

The fitted object with recovered latent process draws added. User-facing `w_samples` are returned by process term; NNGP and DAGAR terms are in original support order. The internal ordered copy used by fitted values and prediction is stored in `w_samples_ordered`. The object also includes `w_samples_stacked` and `recover_iter`. The returned object has class `stLMM_recovery` and can be passed to `fitted()` or `predict()`. For multi-chain input, the returned object has class `stLMM_recovery_chains` and contains `chains`, a list of ordinary `stLMM_recovery` objects. The `sub_sample` rule is applied within each chain.

See Also

`stLMM`, `fitted`, `predict.stLMM_recovery`, `summary.stLMM_recovery`

Examples

```
set.seed(1)
dat <- data.frame(
  y = rnorm(8),
  x = rnorm(8),
  time = seq_len(8)
)

fit <- stLMM(
```

```
  y ~ x + ar1(time),
  data = dat,
  n_samples = 8,
  priors = list(
    resid = list(tau_sq = ig(2, 1)),
    ar1_1 = list(sigma_sq = ig(2, 1), phi = uniform(-0.8, 0.8))
  ),
  warmup = FALSE,
  verbose = FALSE
)
rec <- recover(fit, sub_sample = list(start = 5, thin = 1))
stats::fitted(rec)
```

`resid` *Residual variance formula term*

Description

Defines the Gaussian residual variance model as part of an `stLMM` formula. Omitting the term is equivalent to writing `resid()` or `resid(model = "tau_sq")`: one global residual variance `tau_sq` is estimated.

Usage

```
resid(model = c("tau_sq", "constant", "fixed", "group", "scaled"), group,
  variance, vhat, n = NULL, prior = c("default", "ig", "shannon"),
  method, shape = 4, center = c("mean", "mode"), shrinkage = 10,
  kappa_log_prior = c(mean = 0, sd = 1), tau0_sq_log_prior = NULL,
  starting = NULL, tuning = 0.1)
```

Arguments

`model`	Residual variance model. `"tau_sq"` and `"constant"` estimate one global Gaussian residual variance. `"fixed"` uses fixed row-specific variances. `"group"` estimates one residual variance per group. `"scaled"` estimates a low-dimensional scaling model for supplied row-specific variances.
`group`	Grouping variable for sampled group-specific residual variances.
`variance`	Numeric vector or expression evaluated in the model data giving one residual variance per row. This is used by `model = "fixed"`, `model = "group"` with direct-estimate-informed priors, and `model = "scaled"`. Variances for observed response rows must be finite and positive. Missing response rows may have missing variances because they do not enter the likelihood.
`vhat`	Alias for `variance`, useful when the supplied variances are direct-estimate variances. For grouped direct-estimate-informed residual variances, this is supplied per row and must be constant within group for observed rows.
`n`	Optional effective sample size. For `prior = "shannon"`, this is group-level. For `model = "scaled"`, this is row-level and controls shrinkage toward a common residual scale.

prior	Prior construction for model = "group" when variance is supplied. "ig" uses a constant-shape inverse-gamma prior centered on the supplied variance. "shannon" uses the effective-sample-size construction described below.
method	Alias for prior; retained only to make the argument name read naturally when comparing constant-shape and Shannon-style constructions.
shape	Inverse-gamma shape for prior = "ig".
center	Whether the constant-shape inverse-gamma prior mean or mode is centered on variance.
shrinkage	Positive constant controlling how quickly the sample-size aware scaled model trusts variance. The weight is n / (n + shrinkage).
kappa_log_prior	Mean and standard deviation of the normal prior on log(kappa).
tau0_sq_log_prior	Optional mean and standard deviation of the normal prior on log(tau0_sq) for the sample-size aware scaled model. If omitted, the prior is centered on the median observed variance.
starting	Optional starting values for residual variance parameters used by the grouped direct-estimate-informed and scaled models. For model = "group" without variance, supply starting values through starting = list(resid = list(tau_sq = value)) in stLMM.
tuning	Initial log-scale Metropolis proposal scale for sampled residual variance parameters.

Details

resid() is an stLMM formula term, not the usual fitted-model resid() or residuals() extractor. Only one resid() term is allowed, and it cannot be used in interactions. The formula term selects the residual variance model; corresponding priors, starting values, and tuning values are supplied through the resid block in priors, starting, and tuning. Gaussian residual variance models are not used with Polya-Gamma likelihoods such as family = "binomial" or family = "negative_binomial".

resid(model = "tau_sq") estimates one scalar residual variance. Use priors = list(resid = list(tau_sq = ig(shape, scale))) or priors = list(resid = list(tau_sq = half_t(df, scale))) to place a prior on this variance model.

resid(model = "fixed", variance = vhat) uses observed-row precision obs_precision_i = 1 / vhat_i. When this model is used, tau_sq is not sampled. Prediction of new response draws with y_samples = TRUE requires residual variance information for the prediction rows.

resid(model = "group", group = g) estimates one residual variance per group. Priors are supplied through priors$resid; use priors = list(resid = list(tau_sq = ig(shape, scale))) to apply one prior to all groups, or supply a named list with one prior per group. Starting values and tuning values are supplied through starting$resid$tau_sq and tuning$resid$tau_sq; scalars are recycled over groups and named vectors or lists may be used when group-specific values are needed. Values marked with fixed are held fixed, assigned zero tuning, and do not require residual group priors.

resid(model = "group", group = g, variance = vhat, prior = "ig") also estimates one residual variance per group, but constructs inverse-gamma priors from externally supplied group-level

variance information. For `prior = "ig"`, shape is the inverse-gamma shape and the scale is chosen so either the prior mean or prior mode equals `vhat`. With `prior = "shannon"`, the prior is `IG(n / 2, (n - 1) * vhat / 2)` and requires `n > 1`; larger `n` gives a more concentrated prior around the supplied direct-estimate variance. Starting values for this direct-estimate-informed grouped model are supplied with the `starting` argument inside `resid(...)`.

`resid(model = "scaled", variance = vhat)` estimates a low-dimensional multiplier for direct-estimate variances. Without `n`, the model is `tau_i^2 = kappa * vhat_i`. With `n`, the model uses `log(tau_i^2) = log(kappa) + w_i log(vhat_i) + (1 - w_i) log(tau0_sq)` where `w_i = n_i / (n_i + shrinkage)`. The positive parameters `kappa` and `tau0_sq` are sampled on the log scale, so zero or negative residual variances are not possible. Starting values for these parameters are supplied with the `starting` argument inside `resid(...)`.

Value

An object of class `stLMM_residual` that encodes the residual variance model selected by the formula term. The object is intended for use inside `stLMM` formulas and stores the residual model type plus the user-supplied grouping, variance, starting-value, and tuning information needed when the model frame is built.

See Also

`stLMM`, `predict.stLMM_recovery`

Examples

```
area <- rep(letters[1:3], each = 2)
vhat <- rep(c(0.2, 0.4, 0.6), each = 2)
n_eff <- rep(c(10, 20, 30), each = 2)

resid()
resid(model = "fixed", variance = vhat)
resid(model = "group", group = area)
resid(model = "group", group = area, variance = vhat, prior = "ig", shape = 6)
resid(model = "scaled", variance = vhat, n = n_eff)
```

---

stLMM *Fit a Bayesian mixed model with structured latent processes*

Description

Fits Bayesian linear mixed models with fixed effects, optional explicit iid grouped random effects, and optional structured latent process terms. Structured process terms are integrated out during parameter updates. For Gaussian process models, their latent draws are recovered afterward with `recover`. For Polya-Gamma process models, the in-chain process draws already used by the augmentation step are saved and reused by fitted-value, recovery, and prediction methods.

For `family = "binomial"`, the sampler uses Polya-Gamma latent variables. Conditional on the Polya-Gamma weights, the model is fit as a Gaussian working model with diagonal observation precision equal to the current weights. Polya-Gamma random variates are generated through

BayesLogit's hybrid sampler. Binomial trials default to one per row and can be supplied through `trials`. The Gaussian `resid()` residual variance term and `tau_sq` controls are not used for binomial fits. For structured binomial process models, the sampler saves in-chain process draws by default; `recover` selects from those saved draws.

For `family = "negative_binomial"`, the response must be non-negative integer counts and `size` must be supplied as a positive fixed scalar. The sampler uses the log-mean parameterization $\eta_i = \log(\mu_i)$ and an internal logit tilt $\psi_i = \eta_i - \log(\texttt{size})$. Conditional on the Polya-Gamma weights, the model again becomes a Gaussian working model with diagonal observation precision equal to the current weights. Residual `resid()` terms and `tau_sq` controls are not used for this likelihood. Larger `size` values make the negative-binomial distribution closer to Poisson, but can slow the sampler because the Polya-Gamma shape is the observed count plus `size`.

Usage

```
stLMM(formula, data = parent.frame(), starting = list(), tuning = NULL,
  priors = NULL, family = "gaussian", trials = NULL,
  size = NULL, n_samples, n_omp_threads = 1, nngp_search = c("fast", "brute"),
  verbose = TRUE, n_report = 100, warmup = TRUE, metropolis = list(),
  cholmod_control = list(), save_process = NULL, chains = 1L,
  chain_control = list(), describe_terms = FALSE, ...)
```

Arguments

`formula`	A model formula with fixed effects, optional `offset()` terms, optional iid grouped random-effect terms such as `iid(group)` or `x:iid(group)`, and optional structured process terms such as `ar1()`, `gp()`, `nngp()`, `car()`, `dagar()`, `car_time()`, or `dagar_time()`.
`data`	A data frame or environment containing variables in `formula`.
`starting`	Optional named list of starting values. The public convention uses underscores. Default residual variance is supplied as `starting = list(resid = list(tau_sq = value))`. Process starts use term-specific blocks, for example `starting = list(nngp_1 = c(sigma_sq = 1, phi = 1))`. IID random-effect variance starts use the same term-block convention, for example `starting = list(iid_1 = list(sigma_sq = 1))`. Covariance and residual variance parameters can be fixed with `fixed`, for example `starting = list(resid = list(tau_sq = fixed(0.5)))`.
`tuning`	Optional named list of initial Metropolis proposal scales for covariance and residual variance parameters. A zero tuning value fixes the corresponding parameter. Parameters supplied with `fixed` are assigned zero tuning automatically.
`priors`	Named list of prior objects created with constructors such as `flat`, `normal`, `ig`, `uniform`, `log_normal`, `gamma_dist`, `half_normal`, `half_t`, and `beta_dist`. Fixed effects use `priors = list(beta = flat())` or `priors = list(beta = normal(mean, sd))`. Default residual variance uses `resid$tau_sq`; for example, `priors = list(resid = list(tau_sq = ig(shape, scale)))` puts an inverse-gamma prior directly on `tau_sq`, while `priors = list(resid = list(tau_sq = half_t(df, scale)))` defines a half-t prior on the residual standard deviation $\sqrt{\tau^2}$ and transforms it internally to the variance scale. Process terms require term-specific blocks, for example `priors = list(nngp_1 = list(sigma_sq = ig(shape, scale),`

phi = uniform(lower, upper))). IID random-effect variance priors use the same term-block convention, for example priors = list(iid_1 = list(sigma_sq = ig(shape, scale))). Priors are not required for covariance or residual variance parameters supplied with fixed.

family — Model likelihood family. Use "gaussian" for Gaussian linear mixed models or "binomial" for binomial logistic models using Polya-Gamma Gibbs updates. Use "negative_binomial" for fixed-size negative-binomial log-mean count models using Polya-Gamma Gibbs updates. The default is "gaussian". For compatibility with familiar R model syntax, gaussian() and binomial() are also accepted.

trials — Optional binomial trial counts for family = "binomial". The default NULL uses one trial per row, i.e. Bernoulli data. A single character string names a column in data; otherwise supply a positive integer-valued vector with length matching the model frame. The response should contain successes and must be between zero and trials.

size — Positive fixed negative-binomial size parameter for family = "negative_binomial". The fitted linear predictor is $\eta_i = \log(\mu_i)$, where μ_i is the mean count.

n_samples — Number of MCMC samples.

n_omp_threads — Number of OpenMP threads for supported kernels.

nngp_search — Nearest-neighbor search used when constructing NNGP graphs. "fast" uses the exact history-restricted k-d tree implementation. "brute" checks every eligible historical node and is mainly useful for testing and debugging.

verbose — Logical; print setup, sampler-start, and progress reports.

n_report — Progress report interval for retained sampling and warmup. For warmup, reports are rounded to completed warmup batches. Use 0 to suppress progress reports while keeping other verbose setup messages.

warmup — Logical or list controlling discarded Metropolis proposal-scale calibration before retained sampling. TRUE uses default calibration; FALSE disables it. A list may contain enabled, batch_length, min_batches, max_batches, target, and near_zero. min_batches forces at least that many warmup batches before early stopping is allowed; max_batches remains the ceiling. Defaults are enabled = TRUE, batch_length = 25, min_batches = 0, max_batches = 20, and near_zero = 0.02. When target is omitted, the default acceptance interval is c(0.15, 0.45) for block updates and c(0.30, 0.60) for scalar updates.

metropolis — Optional sampler-control list for Metropolis updates of covariance and residual variance parameters. The default list(blocking = "joint") keeps all active parameters in one adaptive covariance block. Built-in alternatives are "by_term", "residual_process", "variance_theta", "process_variance", and "scalar". A single string such as metropolis = "by_term" is also accepted. A list may also contain target_accept for retained-sampling adaptation and batch_length for retained adaptation batches. Defaults are target_accept = 0.234 for block updates, 0.44 for scalar updates, and batch_length = 25.

cholmod_control — Optional sparse factorization control list. The ordering element controls the CHOLMOD fill-reducing ordering used for latent sparse matrix factorizations.

Supported values are "auto" (the default CHOLMOD strategy), "best" (try CHOLMOD's full built-in method set), "natural", "amd", "metis", "nesdis", and "colamd". A single string such as cholmod_control = "amd" is also accepted. The postorder element is logical and defaults to TRUE. METIS-based choices require the CHOLMOD library exposed by **Matrix** to have been built with METIS/partitioning support.

save_process — Controls whether structured latent process draws are saved during MCMC. Use NULL for the default: process draws are saved automatically for Polya-Gamma process models as list(start = 1, thin = 1) and not saved otherwise. Use TRUE as shorthand for list(start = 1, thin = 1), use FALSE to turn saving off, or use a list with start and thin, for example save_process = list(start = 501, thin = 5). For Polya-Gamma process models, saved in-chain process draws are required for recover(), fitted(), and process prediction. Enabling save_process is only supported for Polya-Gamma models with structured process terms.

chains — Positive integer number of MCMC chains to run. The default 1L returns an ordinary stLMM object. Values greater than one run sequential chains and return an stLMM_chains object.

chain_control — Optional list controlling multi-chain initialization. seed initializes reproducible chain-specific random number streams when supplied. dispersion controls log-scale jitter for omitted positive starting values.

describe_terms — Logical; print model term diagnostics before sampling.

... — Currently unused.

Details

Predictor variables must be complete. Missing numeric responses are allowed: only observed response rows enter the likelihood, while the full row support is retained for fitted values, recovery, and prediction metadata. The number of missing responses is reported by print() and summary(). Formula offsets are supported through standard R offset() terms. The offset is a known additive component of the linear predictor. For negative-binomial count models, offset(log_exposure) gives $\log(\mu_i) = \log(\text{exposure}_i) + \cdots$; for binomial models it is a known log-odds contribution. Fitted values and predictions include the offset on both link and response scales. When predicting with newdata, variables used inside offset() must be present in newdata.

The intended workflow is:

```
fit <- stLMM(...)
summary(fit)
rec <- recover(fit)
stats::fitted(rec)
predict(rec, newdata = ...)
```

For models without structured process terms, fitted() and predict() can be called directly on the fit object.

Prior lists use explicit prior constructors. Raw numeric vectors such as c(2, 1) are not accepted as priors. Common patterns are:

```
## residual variance
priors = list(resid = list(tau_sq = ig(shape, scale)))
priors = list(resid = list(tau_sq = half_t(df = 3, scale = 0.5)))

## fixed effects
priors = list(beta = flat())
priors = list(beta = normal(mean = 0, sd = 10))

## known additive offset
y ~ x + offset(log_exposure) + car(area, graph = g)

## fixed direct-estimate sampling variances
y ~ x + resid(model = "fixed", variance = vhat)

## sampled group-specific residual variances
y ~ x + resid(model = "group", group = area)
priors = list(resid = list(tau_sq = half_t(df = 3, scale = 1)))

## direct-estimate-informed group-specific residual variances
y ~ x + resid(model = "group", group = area, variance = vhat,
              prior = "ig", shape = 6)

## scaled direct-estimate sampling variances
y ~ x + resid(model = "scaled", variance = vhat, n = n_eff)

## iid random-effect variance for iid(group)
priors = list(iid_1 = list(sigma_sq = ig(shape, scale)))

## fixed covariance parameters
starting = list(
  resid = list(tau_sq = fixed(0.5)),
  nngp_1 = list(sigma_sq = fixed(1), phi = fixed(3))
)

## process covariance controls
priors = list(nngp_1 = list(sigma_sq = ig(shape, scale),
                            phi = uniform(lower, upper)))

## CAR controls
priors = list(car_1 = list(sigma_sq = ig(shape, scale),
                           rho = uniform(lower, upper)))

## DAGAR controls
priors = list(dagar_1 = list(sigma_sq = ig(shape, scale),
                             rho = uniform(lower, upper)))

## separable CAR-time controls
priors = list(car_time_1 = list(sigma_sq = ig(shape, scale),
```

```
                                rho = uniform(lower, upper),
                                phi = uniform(lower, upper)))

## separable CAR-time controls with time_model = "exp"
priors = list(car_time_1 = list(sigma_sq = ig(shape, scale),
                                rho = uniform(lower, upper),
                                lambda = uniform(lower, upper)))

## separable DAGAR-time controls
priors = list(dagar_time_1 = list(sigma_sq = ig(shape, scale),
                                  rho = uniform(lower, upper),
                                  phi = uniform(lower, upper)))
```

- **Starting values, tuning, and fixed parameters.** Omitting `starting` uses data-based or conservative defaults. Omitting `tuning` uses default initial Metropolis proposal scales. Setting a tuning value to zero fixes that covariance or residual variance parameter. Parameters marked with `fixed()` use their supplied values and are omitted from Metropolis proposal blocks.
- **Priors and parameter transforms.** Process terms require priors for parameters that are sampled. The Gaussian likelihood defaults to a flat fixed effect prior, while Polya-Gamma likelihoods default to `normal(mean = 0, sd = 10)` for fixed effects. The `flat()` and `normal()` constructors are accepted only for `priors$beta`. Variance parameters are positive. The `half_normal()` and `half_t()` variance-prior constructors are specified on the corresponding standard deviation scale and transformed internally to the variance scale. Covariance theta parameters use a bounded-logit proposal transform and therefore require finite support. For positive theta parameters, `log_normal()` and `gamma_dist()` must include `support = c(lower, upper)`. See `stLMM_priors` and `stLMM_terms` for details.
- **Multiple chains.** For `chains > 1`, omitted starting values are dispersed automatically from the defaults. With `chain_control$seed`, chain-specific random number streams are generated from the supplied seed, so dispersed starts are reproducible across different Metropolis blocking choices. If a starting value is supplied by the user, it controls that parameter: scalar values are recycled to all chains, and vectors with length `chains` provide chain-specific starts. Other lengths are rejected. For process terms, chain-specific starts should be supplied in list blocks, for example `starting = list(nngp_1 = list(sigma_sq = c(1, 2, 3), phi = c(0.2, 0.4, 0.6)))`.
- **Warmup and retained-sampling adaptation.** By default, the Metropolis sampler runs a short discarded warmup phase that calibrates proposal scales before retained samples are collected. Warmup draws are not returned in posterior sample matrices; warmup diagnostics are stored in `fit$adaptive_metropolis$warmup`. Retained-sampling adaptation is controlled separately through `metropolis$target_accept` and `metropolis$batch_length`.
- **Metropolis blocking.** Metropolis blocking changes only the proposal partition used for covariance and residual variance parameters; it does not change the model. Blocking can improve mixing for multi-term or residual-variance models, but each block may require a separate collapsed likelihood evaluation and sparse factorization. The default `"joint"` block is usually the fastest choice. Treat the other blocking schemes as diagnostic or model-specific alternatives: compare trace plots, R-hat, effective sample sizes, and runtime before changing the default for routine analyses. `metropolis = "scalar"` updates one active covariance or residual variance parameter at a time using scalar random-walk adaptation. It can be useful

when block proposals mix poorly, but it may be substantially slower for models with expensive collapsed likelihood evaluations.

- **Sparse factorization controls.** The `cholmod_control` argument is intended for advanced diagnostics and performance benchmarking. The default `ordering = "auto"` uses CHOLMOD's default ordering strategy. Alternative orderings can change sparse Cholesky fill-in, floating-point work, and runtime without changing the statistical model. Use `describe_terms = TRUE` or inspect `fit$term_description` to compare the selected CHOLMOD ordering, fill ratio, `lnz`, and flops. The `"best"` option can make symbolic analysis slower because CHOLMOD tries more orderings before factorization.
- **Fixed residual variances.** When `resid(model = "fixed", variance = vhat)` is used, `tau_sq` is not sampled and should not be supplied in `starting`, `tuning`, or `priors`.

See `stLMM_terms` for the package-native formula terms recognized by `stLMM()`.

Value

With `chains = 1`, an object of class `stLMM`. With `chains > 1`, an object of class `stLMM_chains` with component `chains`, a list of ordinary `stLMM` objects, one per chain.

For each ordinary fit, posterior samples are available both as top-level entries such as `beta_samples`, `tau_sq_samples`, `sigma_sq_samples`, `theta_samples`, and `alpha_samples`, and through the active-component list `samples`. Empty sample blocks for absent model components are `NULL`. Covariance Metropolis diagnostics are returned as `covariance_acceptance` and `adaptive_metropolis`. The default is one global covariance block; optional blocking and scalar-update diagnostics are stored in `adaptive_metropolis$blocks`. The object also contains `timing`, with CPU and elapsed wall time for the compiled sampler call, `term_description`, and the stored backend used by recovery and prediction. Multi-chain fits also include top-level sampler timing by chain and total sampler timing across chains.

Latent process draws are saved during fitting only for Polya-Gamma likelihoods with structured process terms. The default in that case is `save_process = list(start = 1, thin = 1)`, because those in-chain process draws are already needed by the Polya-Gamma augmentation and post-fit Polya-Gamma process reconstruction is not used. For Gaussian process models, the sampler remains fully collapsed with respect to the process terms and `recover()` is the intended post-fit step for drawing latent process samples. Multi-chain fits can be converted to a `coda::mcmc.list` with `as_mcmc` and summarized with `summary()`, which includes chain diagnostics when possible.

See Also

`stLMM_terms`, `stLMM_priors`, `recover`, `fitted`, `predict.stLMM_recovery`, `summary.stLMM`

Examples

```
set.seed(1)
dat <- data.frame(
  y = rnorm(8),
  x = rnorm(8),
  time = seq_len(8)
)

fit <- stLMM(
```

```
  y ~ x + ar1(time),
  data = dat,
  n_samples = 8,
  priors = list(
    resid = list(tau_sq = ig(2, 1)),
    ar1_1 = list(sigma_sq = ig(2, 1), phi = uniform(-0.8, 0.8))
  ),
  warmup = FALSE,
  verbose = FALSE
)
rec <- recover(fit, sub_sample = list(start = 5, thin = 1))
stats::fitted(rec)
```

---

stLMM-methods *Print, summarize, and plot stLMM objects*

Description

S3 methods for compact inspection of fitted, recovered, and prediction objects. Summaries report posterior means, standard deviations, and quantiles for active sample blocks only. Plot methods use base graphics and choose conservative defaults to avoid drawing very large posterior or prediction objects by accident.

Usage

```
## S3 method for class 'stLMM'
print(x, ...)
## S3 method for class 'stLMM'
summary(object, probs = c(0.025, 0.5, 0.975),
  burn = 0L, parameters = NULL, ...)
## S3 method for class 'stLMM'
plot(x, type = c("trace", "density"), parameters = NULL,
  max_parameters = 12L, burnin = 0L, thin = 1L, ...)
## S3 method for class 'stLMM_chains'
print(x, ...)
## S3 method for class 'stLMM_chains'
summary(object, probs = c(0.025, 0.5, 0.975),
  diagnostics = TRUE, include_w = FALSE, burn = 0L,
  parameters = NULL, ...)
## S3 method for class 'stLMM_chains'
plot(x, type = c("trace", "density"),
  parameters = NULL, max_parameters = 12L, include_w = FALSE,
  n_col = NULL, chain_colors = NULL, lwd = 1, burnin = 0L,
  thin = 1L, ...)
## S3 method for class 'stLMM_recovery'
print(x, ...)
## S3 method for class 'stLMM_recovery'
summary(object, probs = c(0.025, 0.5, 0.975),
```

```
  burn = 0L, parameters = NULL, include_w = FALSE, max_w = 20L, ...)
## S3 method for class 'stLMM_recovery_chains'
summary(object,
  probs = c(0.025, 0.5, 0.975), diagnostics = TRUE,
  include_w = FALSE, burn = 0L, parameters = NULL, ...)
## S3 method for class 'stLMM_recovery'
plot(x, term = NULL, nodes = NULL,
  type = c("trace", "density", "fitted"), observed = NULL,
  max_nodes = 12L, burnin = 0L, thin = 1L, ...)
## S3 method for class 'stLMM_recovery_chains'
plot(x, type = c("trace", "density"),
  parameters = NULL, max_parameters = 12L, include_w = FALSE, ...)
## S3 method for class 'stLMM_prediction'
print(x, ...)
## S3 method for class 'stLMM_prediction'
summary(object, probs = c(0.025, 0.5, 0.975),
  include_y = !is.null(object$y_samples), ...)
## S3 method for class 'stLMM_prediction'
plot(x, sample = c("mu", "y"),
  type = c("interval", "density", "scatter"), rows = NULL,
  observed = NULL, probs = c(0.025, 0.5, 0.975), max_rows = 200L,
  burnin = 0L, thin = 1L, ...)
## S3 method for class 'stLMM_prediction_chains'
plot(x, sample = c("mu", "y"),
  type = c("interval", "density"), rows = NULL,
  probs = c(0.025, 0.5, 0.975), max_rows = 200L,
  burnin = 0L, thin = 1L, ...)
```

Arguments

x, object	An stLMM, stLMM_recovery, or stLMM_prediction object, or the corresponding multi-chain object.
probs	Posterior quantiles to report. Prediction interval plots expect three probabilities: lower, center, and upper.
type	Plot type. For fit objects, "trace" or "density". For recovery objects, "trace", "density", or "fitted". For prediction objects, "interval", "density", or "scatter".
parameters	Optional exact parameter names to summarize or plot for an stLMM fit. For summaries, unlisted parameters are omitted from the returned parameter table or tables.
max_parameters	Maximum number of parameters to plot by default.
burn	Number of initial posterior draws to discard before computing fit summaries and multi-chain diagnostics. The default 0L uses all retained draws. This affects only the summary call; it does not modify the stored posterior samples.
diagnostics	Logical; for multi-chain summaries, compute **coda** diagnostics such as potential scale reduction and effective sample size.

burnin	Number of initial plotted draws to discard. This affects only the plotting call; it does not modify the stored posterior samples.
thin	Keep every thin-th plotted draw after burnin. This affects only the plotting call.
include_w	Logical; include summaries of saved or recovered latent process samples. Defaults to FALSE because these can be large.
n_col	Optional number of columns for multi-chain fit trace or density plot panels. Defaults to a compact base graphics layout.
chain_colors	Optional vector of colors for multi-chain fit plots. Values are recycled to the number of chains.
lwd	Line width for multi-chain fit trace or density plots.
max_w	Maximum number of latent process nodes summarized when include_w = TRUE.
term	Recovered process term to plot. Defaults to the first recovered term.
nodes	Latent process nodes to plot.
max_nodes	Maximum number of recovered nodes to plot by default.
include_y	Logical; include posterior predictive observation summaries when y_samples were simulated.
sample	Prediction sample matrix to plot: "mu" or "y".
rows	Prediction rows to plot.
observed	Observed values for recovery fitted-value scatter plots or prediction scatter plots. Recovery plots use the fitted response by default.
max_rows	Maximum number of prediction rows plotted by default.
...	Additional arguments passed to base plotting functions.

Value

Print methods return x invisibly. Summary methods return summary objects. Plot methods return x invisibly.

---

stLMM-priors *Prior constructors for stLMM*

Description

Constructors for prior and fixed-parameter objects used by stLMM.

Usage

```
flat()

normal(mean = 0, sd = 1)

ig(shape, scale)

uniform(lower, upper)

log_normal(meanlog, sdlog, support = NULL)

gamma_dist(shape, rate, support = NULL)

half_normal(scale)

half_t(df, scale)

beta_dist(shape1, shape2)

fixed(value)
```

Arguments

mean, sd	Mean and standard deviation for independent normal fixed-effect priors created by `normal()`. Scalars are recycled across fixed effects; vectors must match the number of fixed effects.
shape, scale	For `ig()`, positive inverse-gamma shape and scale parameters. For `half_normal()`, positive standard-deviation scale.
lower, upper	Finite support limits for `uniform()`.
meanlog, sdlog	Mean and standard deviation on the log scale for `log_normal()`; `sdlog` must be positive.
support	Optional finite support `c(lower, upper)`. This is required when `log_normal()` or `gamma_dist()` is used for a covariance theta parameter.
rate	Positive gamma rate parameter.
df	Positive degrees of freedom for `half_t()`.
shape1, shape2	Positive beta shape parameters.
value	Finite numeric scalar value at which a covariance or variance parameter should be fixed.

Details

Raw numeric vectors such as `c(2, 1)` are not accepted as priors. Use an explicit constructor so the prior family and parameter support are clear.

`fixed(value)` marks a covariance or variance parameter as fixed at `value`. It is used in `starting`, not `priors`. Internally this sets the starting value and sets the corresponding Metropolis tuning value to zero. Fixed parameters are omitted from Metropolis proposal blocks and do not require

priors. Their values still define the likelihood and latent-process precision. If a fixed parameter also has an explicitly positive tuning value, `stLMM()` errors.

`flat()` and `normal(mean, sd)` are fixed-effect priors and are currently accepted only as `priors = list(beta = ...)`. A flat beta prior is the improper prior used by earlier versions of the sampler. A normal beta prior is an independent Gaussian prior on the fixed-effect coefficients. The Gaussian likelihood defaults to `flat()`; Polya-Gamma likelihoods default to `normal(mean = 0, sd = 10)`.

The `_dist` suffix is used only where the natural constructor name would conflict with an existing R function or a common model parameter name, as in `gamma_dist()` and `beta_dist()`.

The inverse-gamma constructor uses the shape/scale parameterization

$$p(x \mid a, b) = \frac{b^a}{\Gamma(a)} x^{-(a+1)} \exp\{-b/x\}, \quad x > 0,$$

where `shape = a` and `scale = b`. Thus the second argument to `ig()` is a scale parameter for the inverse-gamma distribution, not a rate.

Other positive variance priors use the following parameterizations. The `log_normal(meanlog, sdlog)` prior is the usual log-normal density with `meanlog` and `sdlog` on $\log(x)$. The `gamma_dist(shape, rate)` prior uses the gamma shape/rate density $p(x) \propto x^{a-1} \exp\{-rx\}$. The `half_normal(scale)` and `half_t(df, scale)` priors are specified on the standard deviation $s = \sqrt{x}$ and transformed internally to the variance scale by the Jacobian $1/(2\sqrt{x})$. Up to normalizing constants, this gives $p(x) \propto x^{-1/2} \exp\{-x/(2A^2)\}$ for the half-normal and $p(x) \propto x^{-1/2}\{1 + x/(\nu A^2)\}^{-(\nu+1)/2}$ for the half-t.

`beta_dist(shape1, shape2)` uses the standard beta density $p(x) \propto x^{a-1}(1-x)^{b-1}$ on $0 < x < 1$.

Variance parameters, including `tau_sq` and process `sigma_sq`, are strictly positive. Sampled residual and process variances may use `ig()`, `log_normal()`, `gamma_dist()`, `half_normal()`, or `half_t()`. The half-normal and half-t priors are defined on the standard deviation scale and transformed internally to the variance scale. IID grouped random-effect variances currently use conjugate Gibbs updates and must use `ig()`.

When `resid(model = "group", group = group)` is used, the grouped residual variance priors are supplied through `priors$resid`. A single prior can be recycled over groups, for example `priors = list(resid = list(tau_sq = half_t(df = 3, scale = 1)))`. Alternatively, `priors$resid` may be a named list with one prior for each observed residual group.

Covariance theta parameters use a bounded-logit proposal transform. Therefore, all theta priors must have finite support. Positive theta parameters such as `phi`, `lambda`, `nu`, `a`, `c`, and Gneiting `delta` when free may use `uniform()`, `log_normal(..., support = c(lower, upper))`, or `gamma_dist(..., support = c(lower, upper))`. Unit-interval theta parameters such as mixture `alpha` and Gneiting `alpha`, `beta`, and `gamma` may use `uniform()` or `beta_dist()`. Bounded AR1 parameters currently use `uniform()`.

For exponential covariance $\exp(-\phi h)$, the distance where correlation equals 0.05 is approximately $-\log(0.05)/\phi$. This is often a useful way to choose finite support for `phi` after coordinates have been scaled or projected into meaningful units. For binomial and other weak-information settings, spatial decay parameters can be weakly identified and may be sensitive to the chosen support, neighbor size, and coordinate scale; inspect trace plots and posterior mass near prior boundaries.

Value

Prior constructors return objects of class `stLMM_prior` describing the prior family, parameters, support, and scale used by `stLMM`. `fixed()` returns an object of class `stLMM_fixed_parameter`

marking a covariance or variance parameter as fixed at the supplied numeric value.

See Also

stLMM, stLMM_terms, get_cor_models

Examples

```
priors <- list(
  beta = normal(mean = 0, sd = 10),
  resid = list(tau_sq = ig(2, 0.1)),
  nngp_1 = list(
    sigma_sq = half_t(df = 4, scale = 1),
    phi = log_normal(log(5), 0.75, support = c(0.5, 20)),
    nu = uniform(0.1, 2)
  )
)

starting <- list(
  resid = list(tau_sq = fixed(0.5)),
  nngp_1 = list(sigma_sq = fixed(1), phi = fixed(3))
)
```

---

stLMM-terms *stLMM formula terms*

Description

Formula-term constructors recognized by stLMM.

Details

The symbols documented here are package-native model terms used inside an stLMM() formula. They are not exported as ordinary R functions to call outside a model formula.

IID grouped random effects

iid(group) adds an iid Gaussian random intercept for the observed levels of group. It contributes one column per observed group level to the explicit random-effect design matrix Z.

x:iid(group) adds an iid Gaussian group-varying slope for the numeric covariate x. It contributes one column per observed group level, with row entries scaled by x.

iid(group) + x:iid(group) creates two separate scalar iid random-effect terms with separate variance parameters, typically named iid_1 and iid_2. This is not a correlated intercept/slope block.

IID random effects are sampled through the explicit Z alpha random-effect path. They do not require recover. Prediction from a fit object is available for existing grouping levels. New grouping levels in newdata currently error.

Structured latent process terms

`ar1(index)` adds a one-dimensional AR1 latent process over the sorted unique values of `index`. Repeated observations at the same index share a latent process node. The implemented AR1 support uses adjacency in the sorted unique fitted support.

`gp(coord1, coord2, ..., cov_model = "exp")` adds a dense Gaussian process over unique coordinate rows. Spatial covariance models include `cov_model = "exp"` with parameter `phi` and `cov_model = "matern"` with parameters `phi` and `nu`.

`nngp(coord1, coord2, ..., m = 15, ordering = "coord", st_scale = 1, cov_model = "exp")` adds a nearest-neighbor Gaussian process over unique coordinate rows. Supported `ordering` values are `"coord"`, `"default"`, `"maxmin"`, `"hilbert"`, `"random"`, or a numeric permutation. The `"maxmin"` ordering can be very slow for large numbers of unique coordinate rows because it uses an exact iterative max-min construction. The `"hilbert"` ordering uses only the first two coordinate columns to rank nodes; covariance distances still use all coordinate columns according to the selected `cov_model`. For space-time covariance models, `"hilbert"` is currently rejected because the implemented Hilbert ordering is two-dimensional. Space-time covariance models include `cov_model = "sep_exp"`, `"multi_res_sep_exp"`, and `"gneiting"`.

`car(area_id, graph = g, car_model = "proper")` adds a CAR latent process over areal nodes defined by `g`, where `g` is produced by `car_graph`. Rows with the same `area_id` share a latent process node. Areas in the graph that do not have observed responses remain in the latent support and can be recovered when they are represented by rows with missing response values. Prediction is implemented for existing graph areas. New CAR areas currently error. For polygon inputs, `car_graph()` can optionally add auditable nearest-neighbor bridge edges for disconnected island components. `car_model = "proper"` uses the package's original degree-scaled proper CAR precision. `car_model = "leroux"` uses the Leroux CAR precision, which bridges iid area effects and intrinsic-CAR smoothing.

`dagar(area_id, graph = g, ordering = "coord")` adds an ordered directed acyclic graph autoregressive (DAGAR) latent process over the areal nodes in `g`. The same `car_graph` object used by `car()` supplies the undirected adjacency graph; `ordering` orients each adjacency edge from the earlier area to the later area. Supported orderings are `"coord"`, `"default"`, `"maxmin"`, `"hilbert"`, `"random"`, or a numeric permutation. Coordinate-based orderings require an `sf`-based graph; for adjacency-matrix graphs, `"coord"` and `"default"` use the supplied row order. Prediction is implemented for existing graph areas. New DAGAR areas currently error.

`car_time(area_id, time, graph = g, time_model = "ar1", car_model = "proper")` adds a separable CAR-by-time latent process over the full Cartesian product of graph areas and sorted unique fitted time values. The latent support is stacked location-major, so all time points for the first area come first. The current precision is `sigma_sq^-1 * (Q_space %x% Q_time)`, with one variance term for the whole space-time process. The default temporal model is `time_model = "ar1"`, which uses adjacency in the sorted fitted time support. `time_model = "exp"` uses `Corr(t_i, t_j) = exp(-lambda * abs(t_i - t_j))`; fitted time values must be numeric and finite. New prediction rows must use existing areas and existing fitted time values.

`dagar_time(area_id, time, graph = g, ordering = "coord", time_model = "ar1")` adds a separable DAGAR-by-time latent process over the full product of ordered DAGAR graph areas and sorted unique fitted time values. The undirected graph supplied by `g` is first oriented by `ordering` as in `dagar()`, and the resulting ordered DAGAR spatial precision is crossed with the temporal precision. The latent support is stacked ordered-area-major, so all time points for the first ordered area come first. The user-facing recovered samples are returned in the original graph-area order crossed

with fitted time order. The default temporal model is `time_model = "ar1"`. `time_model = "exp"` uses `Corr(t_i, t_j) = exp(-lambda * abs(t_i - t_j))`; fitted time values must be numeric and finite. New prediction rows must use existing graph areas and existing fitted time values.

For space-time covariance models, coordinate order matters and the final coordinate is treated as time. For example, `nngp(lon, lat, time, cov_model = "sep_exp")` is not equivalent to `nngp(time, lon, lat, cov_model = "sep_exp")`.

For space-time NNGP terms, `st_scale` controls only the search geometry used to order nodes and choose nearest neighbors. The search distance is equivalent to Euclidean distance after replacing the final coordinate by `st_scale * time`. Thus `st_scale = 1` uses the coordinates as supplied, while `st_scale = s` treats one unit of the final coordinate as `s` spatial-coordinate units for graph construction. The original coordinates are still stored on the fitted graph and used for covariance evaluation, so `st_scale` does not change covariance parameter interpretation or prior calibration. Prediction from recovered NNGP fits inherits the fitted graph's `st_scale` by default and can override it via `predict.stLMM_recovery`.

Process terms can be covariate-scaled with the same colon convention used for spatially varying coefficients, for example `x:nngp(lon, lat)` or `x:ar1(day)`. `x:car(area_id, graph = g)` is the CAR spatially varying coefficient form. `x:car_time(area_id, time, graph = g)` is the CAR-time spatial-temporal varying coefficient form. `x:dagar_time(area_id, time, graph = g)` is the DAGAR-time spatial-temporal varying coefficient form.

Structured process terms are collapsed during parameter updates. Gaussian process fits use `recover` after fitting to draw latent process samples before using process contributions in `fitted` or `predict.stLMM_recovery`. Polya-Gamma process fits save in-chain process draws by default as `save_process = list(start = 1, thin = 1)` and `recover` selects from those draws.

Covariance functions

Dense GP and NNGP terms use the process variance `sigma_sq` and the theta parameters named by `get_cor_models`.

For `cov_model = "exp"`, with spatial distance h,

$$C(h) = \sigma^2 \exp(-\phi h).$$

For `cov_model = "matern"`,

$$C(h) = \sigma^2 \frac{(\phi h)^\nu}{2^{\nu-1}\Gamma(\nu)} K_\nu(\phi h),$$

where $K_\nu(\cdot)$ is the modified Bessel function of the second kind.

For `cov_model = "sep_exp"`, with spatial distance h and temporal distance u,

$$C(h, u) = \sigma^2 \exp(-\phi h) \exp(-\lambda u).$$

For `cov_model = "multi_res_sep_exp"`,

$$C(h, u) = \sigma^2 \left[\alpha \exp(-\phi_1 h) \exp(-\lambda_1 u) + (1 - \alpha) \exp(-\phi_2 h) \exp(-\lambda_2 u)\right].$$

This two-scale separable covariance follows the spatial-temporal forest inventory model used by May and Finley (2025).

For `cov_model = "gneiting"`,

$$C(h,u) = \sigma^2(1 + a|u|^{2\alpha})^{-(\delta+d/2)} \exp\left\{-\frac{ch^{2\gamma}}{(1+a|u|^{2\alpha})^{\beta\gamma}}\right\}.$$

Here h is Euclidean spatial distance, u is temporal distance, and d is the number of spatial coordinate columns before the final time coordinate. This is Gneiting (2002, equation 12). Parameter supports are $a > 0$, $c > 0$, $0 < \alpha \leq 1$, $0 \leq \beta \leq 1$, $0 < \gamma \leq 1$, and $\delta \geq 0$. Free parameters use the strict interior of these supports where needed by the Metropolis transformation; boundary values such as `beta = fixed(0)`, `alpha = fixed(1)`, `gamma = fixed(1)`, or `delta = fixed(0)` can be fixed explicitly.

The covariance used by Datta et al. (2016, equation 6.1) is recovered in two spatial dimensions by setting `alpha = fixed(1)`, `gamma = fixed(0.5)`, `delta = fixed(0)`, and using `beta` as Datta's interaction parameter. A conservative starting point is to estimate `a`, `c`, and optionally `beta`, while fixing `alpha`, `gamma`, and `delta`.

For `ar1(index)`, the fitted support is the sorted unique values of `index`. The correlation between support positions is

$$\mathrm{Cor}(w_i, w_j) = \phi^{|i-j|}.$$

This is an ordered-support AR1, not a numeric time-gap model.

For `car(area, graph = g)`, the proper CAR precision is

$$Q = \sigma^{-2}(D - \rho G),$$

where G is the binary adjacency matrix stored by `car_graph()` and D is the diagonal degree matrix. The graph is not stored as a row-standardized matrix; rather, this degree-scaled precision is equivalent to a CAR conditional specification whose mean uses the neighbor average,

$$E(w_i \mid w_{-i}) = \frac{\rho}{d_i} \sum_{j \sim i} w_j,$$

with conditional variance σ^2/d_i. The current implementation restricts ρ to $(0, 1)$.

For `car(area, graph = g, car_model = "leroux")`, the Leroux CAR precision is

$$Q = \sigma^{-2}\{(1-\rho)I + \rho(D - G)\}.$$

This model has the same graph input and parameter names as the proper CAR model. As ρ approaches zero, the prior approaches iid area effects with variance σ^2; as ρ approaches one, it approaches an intrinsic-CAR smoothing structure. The implementation keeps ρ strictly inside $(0, 1)$.

For `dagar(area, graph = g)`, let π be the ordering of the graph areas and let

$$N_\pi(i) = \{j : i \sim j,\ \pi^{-1}(j) < \pi^{-1}(i)\}$$

be the directed parent set for area i. Let $n_\pi(i) = |N_\pi(i)|$. The ordered DAGAR conditional specification is

$$w_i \mid w_{<i,\pi} \sim N\left(\frac{\rho}{1 + \{n_\pi(i) - 1\}\rho^2} \sum_{j \in N_\pi(i)} w_j,\ \frac{1 + \{n_\pi(i) - 1\}\rho^2}{1 - \rho^2}\right),$$

where the second normal argument is precision. Nodes with no directed parents have conditional mean zero. In stLMM scale convention,

$$Q = \sigma^{-2} L_\pi' F_\pi L_\pi,$$

where L_π is unit lower triangular and F_π contains the conditional precisions. The ordering is part of the model specification, not a CHOLMOD or display-only ordering. This is the ordered spatial DAGAR model of Datta, Banerjee, Hodges, and Gao (2019).

For `car_time(area, time, graph = g)`, the separable precision is

$$Q = \sigma^{-2}\{Q_{\text{space}}(\rho) \otimes Q_{\text{time}}\}.$$

`car_model` selects whether `Q_space` is the proper CAR precision $D - \rho G$ or the Leroux precision $(1-\rho)I + \rho(D-G)$. With `time_model = "ar1"`, `Q_time` is the ordered-support AR1 precision with parameter `phi`. With `time_model = "exp"`, the temporal correlation is

$$\text{Cor}(w_t, w_{t'}) = \exp(-\lambda|t - t'|),$$

represented internally through its Markov precision on sorted numeric fitted time values.

For `dagar_time(area, time, graph = g)`, the separable precision is

$$Q = \sigma^{-2}\{Q_{\text{dagar}}(\rho) \otimes Q_{\text{time}}\}.$$

`Q_dagar` is the scale-free ordered DAGAR precision defined above, after orienting graph edges by `ordering`. `Q_time` is the same AR1 or exponential-time Markov precision used by `car_time()`. The log determinant is evaluated by the Kronecker identity,

$$\log|Q| = -ST\log(\sigma^2) + T\log|Q_{\text{dagar}}(\rho)| + S\log|Q_{\text{time}}|,$$

where S is the number of graph areas and T is the number of fitted time values.

Controls and priors

Term-specific covariance controls use generated names such as `iid_1`, `ar1_1`, `gp_1`, `nngp_1`, `car_1`, `dagar_1`, `car_time_1`, and `dagar_time_1`. The names are assigned by term type in formula order.

IID variance priors are inverse-gamma prior objects, for example:

```
priors = list(iid_1 = list(sigma_sq = ig(shape, scale)))
```

Process terms require a process variance prior plus finite-support priors for their theta parameters, for example:

```
priors = list(
  nngp_1 = list(
    sigma_sq = ig(shape, scale),
    phi = uniform(lower, upper)
  )
)
```

CAR terms use a process variance prior and CAR `rho` bounds:

```
priors = list(
  car_1 = list(
    sigma_sq = ig(shape, scale),
    rho = uniform(lower, upper)
  )
)
```

DAGAR terms use the same process variance and `rho` prior structure:

```
priors = list(
  dagar_1 = list(
    sigma_sq = ig(shape, scale),
    rho = uniform(lower, upper)
  )
)
```

CAR-time terms use one process variance, spatial CAR `rho`, and a temporal parameter. For `time_model = "ar1"`, the temporal parameter is AR1 `phi`:

```
priors = list(
  car_time_1 = list(
    sigma_sq = ig(shape, scale),
    rho = uniform(lower, upper),
    phi = uniform(lower, upper)
  )
)
```

For `time_model = "exp"`, replace `phi` with positive temporal decay `lambda`.

DAGAR-time terms use the same process variance, ordered-DAGAR `rho`, and temporal parameter structure:

```
priors = list(
  dagar_time_1 = list(
    sigma_sq = ig(shape, scale),
    rho = uniform(lower, upper),
    phi = uniform(lower, upper)
  )
)
```

For `time_model = "exp"`, replace `phi` with positive temporal decay `lambda`.

Use `get_cor_models` to inspect supported `cov_model` values and their correlation parameter names.

References

Gneiting, T. (2002). Nonseparable, stationary covariance functions for space-time data. *Journal of the American Statistical Association*, 97(458), 590–600. doi:10.1198/016214502760047113.

May, P. B. and Finley, A. O. (2025). Spatial-temporal prediction of forest attributes using latent Gaussian models and inventory data. *Spatial Statistics*, 69, 100917. doi:10.1016/j.spasta.2025.100917.

See Also

stLMM, recover, fitted, predict.stLMM_recovery, get_cor_models, stLMM_priors

Examples

```
adj <- matrix(
  c(
    0, 1, 0,
    1, 0, 1,
    0, 1, 0
  ),
  nrow = 3,
  byrow = TRUE,
  dimnames = list(c("a", "b", "c"), c("a", "b", "c"))
)
g <- car_graph(adj)

set.seed(1)
dat <- data.frame(
  y = rnorm(3),
  x = rnorm(3),
  area = c("a", "b", "c")
)

fit <- stLMM(
  y ~ x + car(area, graph = g),
  data = dat,
  priors = list(
    resid = list(tau_sq = ig(2, 1)),
    car_1 = list(sigma_sq = ig(2, 1), rho = uniform(0.05, 0.95))
  ),
  n_samples = 8,
  warmup = FALSE,
  verbose = FALSE
)
rec <- recover(fit)
stats::fitted(rec)
```

---

stunitco *FIA state, unit, and county polygons*

Description

An sf polygon data frame with FIA state, unit, and county codes and names.

Usage

```
stunitco
```

Format

An object of class `sf` inheriting from `data.frame`, with 3,108 rows and 8 columns:

`STATECD` FIA state code.

`STATENM` State name.

`UNITCD` FIA unit code.

`UNITNM` FIA unit name.

`COUNTYCD` County code.

`COUNTYNM` County name.

`COUNTYFIPS` County FIPS code.

`geom` Multipolygon geometry.

The package copy is stored in NAD83 / Conus Albers, EPSG:5070.

Source

Downloaded from the United States Census Bureau on 2019 November 3, originally in Esri Shapefile format from https://www.census.gov/geographies/mapping-files/time-series/geo/carto-boundary-file.html. The original projection was Geographic Coordinate System North American 1983, EPSG:4269. The package copy is projected to EPSG:5070 and omits a small number of very remote island polygons from the original source layer.

See Also

car_graph

Examples

```
data(stunitco)
nrow(stunitco)
names(stunitco)
```

Index