Package 'MetricGraph'

November 10, 2026

Type Package

Title Random Fields on Metric Graphs

Version 1.7.0

Maintainer David Bolin <davidbolin@gmail.com>

Description Facilitates creation and manipulation of metric graphs, such as street or river networks. Further facilitates operations and visualizations of data on metric graphs, and the creation of a large class of random fields and stochastic partial differential equations on such spaces. These random fields can be used for simulation, prediction and inference. In particular, linear mixed effects models including random field components can be fitted to data based on computationally efficient sparse matrix representations. Interfaces to the R packages 'INLA' and 'inlabru' are also provided, which facilitate working with Bayesian statistical models on metric graphs. The main references for the methods are Bolin, Simas and Wallin (2024) <doi:10.3150/23-BEJ1647>, Bolin, Kovacs, Kumar and Simas (2023) <doi:10.1090/mcom/3929> and Bolin, Simas and Wallin (2023) <doi:10.48550/arXiv.2304.03190> a

License GPL (>= 2)

Depends R (>= 4.1.0)

Imports stats, RANN, ggplot2 (>= 3.4.0), igraph (>= 1.3.0), sf (>= 1.0-0), rSPDE (>= 2.7.0), Matrix (>= 1.5-0), methods, Rcpp (>= 1.0.5), R6, lifecycle (>= 1.0.0), sp, dplyr (>= 1.0.0), tidyr (>= 1.1.0), magrittr, broom, zoo, ggnewscale, rlang (>= 0.4.0), foreach, doParallel, spatstat.geom

Suggests knitr, testthat, INLA (>= 24.12.01), inlabru (>= 2.14.0), osmdata, sn, plotly, parallel, optimParallel, numDeriv, SSN2, cowplot, leaflet, mapview, viridis, fmesher (>= 0.6.0), data.table, spatstat.data, digest, withr, curl

Additional_repositories https://inla.r-inla-download.org/R/testing

BugReports https://github.com/davidbolin/MetricGraph/issues

URL https://davidbolin.github.io/MetricGraph/

Copyright The R package and code, and the main programs, were written by and are Copyright by David Bolin, Alexandre B. Simas and Jonas Wallin, and are redistributable under the GNU Public

License, version 2 or later. The package also includes partial codes from another package, which was deprecated in Oct-2023, and whose codes are under the GPL-2 license. For details see the COPYRIGHTS file.

Encoding UTF-8

LazyData true

LazyDataCompression xz

VignetteBuilder knitr

NeedsCompilation yes

LinkingTo Rcpp, RcppEigen

Config/roxygen2/version 8.1.0

Author David Bolin [cre, aut],
Alexandre Simas [aut],
Jonas Wallin [aut]

Repository CRAN

Date/Publication 2026-11-10 20:00:14 UTC

Contents

MetricGraph-package . . . 3
augment.graph_lme . . . 4
bru_mapper.inla_metric_graph_spde . . . 7
cross_validation . . . 7
directional_ou_covariance . . . 9
directional_ou_variance . . . 11
drop_na.metric_graph_data . . . 12
exp_covariance . . . 13
fetch_osm . . . 14
filter.metric_graph_data . . . 15
gg_df.metric_graph_spde_result . . . 16
glance.graph_lme . . . 17
graph_bru_process_data . . . 18
graph_data_spde . . . 18
graph_lgcp_sim . . . 20
graph_lme . . . 22
graph_spde . . . 24
graph_spde_basis . . . 28
graph_spde_make_A . . . 28
graph_starting_values . . . 29
lgcp_graph . . . 30
linnet.to.graph . . . 33
logo_lines . . . 34
make_Q_euler . . . 34
make_Q_spacetime . . . 35
match_mesh_data . . . 35

metric_graph . . . 36
metric_graph_from_osm . . . 70
mutate.metric_graph_data . . . 72
pems . . . 72
pems_repl . . . 73
plot.graph_bru_pred . . . 74
plot.graph_bru_proc_pred . . . 74
posterior_crossvalidation.graph_lme . . . 75
posterior_crossvalidation_loo . . . 76
precompute_lgcp_graph . . . 77
predict.graph_lme . . . 81
predict.inla_metric_graph_spde . . . 83
predict.rspde_metric_graph . . . 85
process_rspde_predictions . . . 87
psp.to.graph . . . 87
sample_spde . . . 88
select.metric_graph_data . . . 89
selected_inv . . . 90
simulate.graph_lme . . . 90
simulate.metric_graph . . . 91
simulate_parallel . . . 93
simulate_spacetime . . . 94
spde_covariance . . . 95
spde_metric_graph_result . . . 96
spde_precision . . . 97
spde_variance . . . 98
stlpp.to.graph . . . 99
summarise.metric_graph_data . . . 99
summary.graph_lme . . . 100
summary.metric_graph . . . 101
summary.metric_graph_spde_result . . . 102
update_graph . . . 102
weighted_stiffness . . . 103

Index **105**

`MetricGraph-package` *Gaussian processes on metric graphs*

Description

'MetricGraph' is used for creation and manipulation of metric graphs, such as street or river networks. It also has several functions thatfacilitates operations and visualizations of data on metric graphs, and the creation of a large class of random fields and stochastic partial differential equations on such spaces. The main models are the Whittle-Matérn fields, which are specified through the fractional elliptic SPDE

$$(\kappa^2 - \Delta)^{\alpha/2}(\tau u(s)) = W,$$

$\kappa, \tau > 0$ and $\alpha > 1/2$ are parameters and W is Gaussian white noise. It contains exact implementations of the above model for $\alpha = 1$ and $\alpha = 2$, and contains approximate implementations, via the finite element method, for any $\alpha > 0.5$. It also implements models based on graph Laplacians and isotropic covariance functions. Several utility functions for specifying graphs, computing likelihoods, performing prediction, simulating processes, and visualizing results on metric graphs are provided. In particular, linear mixed effects models including random field components can be fitted to data based on computationally efficient sparse matrix representations. Interfaces to the R packages 'INLA' and 'inlabru' are also provided, which facilitate working with Bayesian statistical models on metric graphs.

Details

At the heart of the package is the `R6` class `[metric_graph()]`. This is used for specifying metric graphs, and contains various utility functions which are needed for specifying Gaussian processes on such spaces.

Linear mixed effects models are provided (see `[graph_lme]`) and perform predictions (see `[predict.graph_lme]`). The package also has interfaces for 'INLA' (see `[graph_spde]`), and it this interface also works with 'inlabru'.

For a more detailed introduction to the package, see the 'MetricGraph' Vignettes.

Author(s)

Maintainer: David Bolin <davidbolin@gmail.com>

Authors:

- David Bolin <davidbolin@gmail.com>
- Alexandre Simas <alexandre.impa@gmail.com>
- Jonas Wallin <jonas.wallin81@gmail.com>

See Also

Useful links:

- https://davidbolin.github.io/MetricGraph/
- Report bugs at https://github.com/davidbolin/MetricGraph/issues

---

`augment.graph_lme` *Augment data with information from a* `graph_lme` *object*

Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. It includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. It also contains the New columns always begin with a . prefix to avoid overwriting columns in the original dataset.

Usage

```
## S3 method for class 'graph_lme'
augment(
  x,
  newdata = NULL,
  which_repl = NULL,
  sd_post_re = FALSE,
  se_fit = FALSE,
  conf_int = FALSE,
  pred_int = FALSE,
  level = 0.95,
  edge_number = "edge_number",
  distance_on_edge = "distance_on_edge",
  coord_x = "coord_x",
  coord_y = "coord_y",
  data_coords = c("PtE", "spatial"),
  normalized = FALSE,
  no_nugget = FALSE,
  check_euclidean = FALSE,
  ...
)
```

Arguments

x	A graph_lme object.
newdata	A data.frame or a list containing the covariates, the edge number and the distance on edge for the locations to obtain the prediction. If NULL, the fitted values will be given for the original locations where the model was fitted.
which_repl	Which replicates to obtain the prediction. If NULL predictions will be obtained for all replicates. Default is NULL.
sd_post_re	Logical indicating whether or not a .sd_post_re column should be added to the augmented output containing the posterior standard deviations of the random effects.
se_fit	Logical indicating whether or not a .se_fit column should be added to the augmented output containing the standard errors of the fitted values. If TRUE, the posterior standard deviations of the random effects will also be returned.
conf_int	Logical indicating whether or not confidence intervals for the posterior mean of the random effects should be built.
pred_int	Logical indicating whether or not prediction intervals for the fitted values should be built. If TRUE, the confidence intervals for the posterior random effects will also be built.
level	Level of confidence and prediction intervals if they are constructed.
edge_number	Name of the variable that contains the edge number, the default is edge_number.
distance_on_edge	Name of the variable that contains the distance on edge, the default is distance_on_edge.

coord_x	Column (or entry on the list) of the data that contains the x coordinate. If not supplied, the column with name "coord_x" will be chosen. Will not be used if Spoints is not NULL or if data_coords is PtE.
coord_y	Column (or entry on the list) of the data that contains the y coordinate. If not supplied, the column with name "coord_x" will be chosen. Will not be used if Spoints is not NULL or if data_coords is PtE.
data_coords	To be used only if Spoints is NULL. It decides which coordinate system to use. If PtE, the user must provide edge_number and distance_on_edge, otherwise if spatial, the user must provide coord_x and coord_y.
normalized	Are the distances on edges normalized?
no_nugget	Should the prediction be done without nugget?
check_euclidean	Check if the graph used to compute the resistance distance has Euclidean edges? The graph used to compute the resistance distance has the observation locations as vertices.
...	Additional arguments.

Value

A tidyr::tibble() with columns:

- .fitted Fitted or predicted value.
- .relwrconf Lower bound of the confidence interval of the random effects, if conf_int = TRUE
- .reuprconf Upper bound of the confidence interval of the random effects, if conf_int = TRUE
- .fittedlwrpred Lower bound of the prediction interval, if conf_int = TRUE
- .fitteduprpred Upper bound of the prediction interval, if conf_int = TRUE
- .fixed Prediction of the fixed effects.
- .random Prediction of the random effects.
- .resid The ordinary residuals, that is, the difference between observed and fitted values.
- .std_resid The standardized residuals, that is, the ordinary residuals divided by the standard error of the fitted values (by the prediction standard error), if se_fit = TRUE or pred_int = TRUE.
- .se_fit Standard errors of fitted values, if se_fit = TRUE.
- .sd_post_re Standard deviation of the posterior mean of the random effects, if se_fit = TRUE.

See Also

glance.graph_lme

bru_mapper.inla_metric_graph_spde *Metric graph 'inlabru' mapper*

Description

Metric graph 'inlabru' mapper

Usage

```
## S3 method for class 'inla_metric_graph_spde'
bru_get_mapper(model, ...)

## S3 method for class 'bru_mapper_inla_metric_graph_spde'
ibm_n(mapper, ...)

## S3 method for class 'bru_mapper_inla_metric_graph_spde'
ibm_values(mapper, ...)

## S3 method for class 'bru_mapper_inla_metric_graph_spde'
ibm_jacobian(mapper, input, ...)
```

Arguments

model	An inla_metric_graph_spde for which to construct or extract a mapper
...	Arguments passed on to other methods
mapper	A bru_mapper.inla_metric_graph_spde object
input	The values for which to produce a mapping matrix

cross_validation *Perform cross-validation on a list of fitted inlabru models on metric graphs.*

Description

Mirrors rSPDE::cross_validation() for bru fits (output from inlabru::bru()). For each fold, the held-out responses are set to NA with inlabru::bru_set_missing() and the model is re-fit with inlabru::bru_rerun(). When true_CV = FALSE, the original fit's mode$theta is held fixed so only the latent posterior is updated; when TRUE, theta is re-estimated. Posterior response samples are then drawn at the held-out locations via inlabru::generate(). Works with the exact, non-FEM SPDE models in **MetricGraph** (inla_metric_graph_spde) and FEM-based rSPDE models alike. A fit containing an rSPDE component and no exact MetricGraph SPDE is delegated to rSPDE::cross_validation(). Mixed exact/rSPDE model lists use this implementation. Fits without a recognizable package-specific component continue to use the function explicitly called.

Usage

```
cross_validation(
  models,
  model_names = NULL,
  scores = c("mae", "mse", "crps", "scrps", "dss"),
  cv_type = c("k-fold", "loo", "lpo"),
  weight_thr = NULL,
  k = 5,
  percentage = 20,
  number_folds = 10,
  n_samples = 1000,
  return_scores_folds = FALSE,
  orientation_results = c("negative", "positive"),
  include_best = TRUE,
  train_test_indexes = NULL,
  return_train_test = FALSE,
  return_post_samples = FALSE,
  return_true_test_values = FALSE,
  parallelize_RP = FALSE,
  n_cores_RP = parallel::detectCores() - 1,
  true_CV = TRUE,
  save_settings = FALSE,
  model_options_bru = list(),
  print = TRUE,
  fit_verbose = FALSE
)
```

Arguments

models	A fitted model from `inlabru::bru()` or a list of such models. All models must have the same number of likelihoods and be fitted to identical datasets.
model_names	Character vector of model names. Defaults to `names(models)` or `Model 1`, `Model 2`, ... if absent.
scores	Subset of `c("mae", "mse", "crps", "scrps", "dss", "wcrps", "swcrps")`.
cv_type	One of `"k-fold"`, `"loo"`, `"lpo"`.
weight_thr	Threshold for `wcrps` / `swcrps`. Required if either is requested.
k	Number of folds for `k-fold`.
percentage	Train percentage (1-99) for `lpo`.
number_folds	Number of folds for `lpo`.
n_samples	Number of posterior samples for sample-based scoring.
return_scores_folds	If `TRUE`, return per-fold score matrices.
orientation_results	One of `"negative"` (lower is better) or `"positive"` (higher is better).
include_best	Add a `Best` row indicating the best model per score.

`train_test_indexes`
: Optional pre-built fold list. If supplied, `cv_type`, `k`, `percentage`, `number_folds` are ignored.

`return_train_test`
: Return the train/test indices used.

`return_post_samples`
: Return posterior response samples (forces `return_scores_folds = TRUE`).

`return_true_test_values`
: Return the true response values at test points.

`parallelize_RP`
: Parallelize CRPS/SCRPS pairwise integrals.

`n_cores_RP`
: Number of cores for `parallelize_RP`.

`true_CV`
: If `TRUE`, re-estimate hyperparameters on each training fold; if `FALSE`, update the latent posterior on the training fold with the original hyperparameters fixed.

`save_settings`
: If `TRUE`, return the CV settings used.

`model_options_bru`
: List of options passed to inlabru.

`print`
: Print partial progress.

`fit_verbose`
: Pass `verbose` through to INLA when refitting.

Value

Either a `data.frame` (default), or a list with `scores_df` plus the optional components requested by `return_*` / `save_settings` arguments.

---

directional_ou_covariance

Closed-form covariance of the directional OU model

Description

Evaluates the covariance function of the "proper global OU" directional model (Whittle-Matern alpha = 1 with directional vertex conditions) in closed form, via the last-common-ancestor (LCA) representation, without assembling or inverting the precision matrix.

Usage

```
directional_ou_covariance(
  graph,
  kappa,
  tau,
  PtE = NULL,
  PtE2 = NULL,
  sigma_source = NULL,
  normalized = TRUE,
  cpp = TRUE
)
```

Arguments

graph	A metric_graph object with directional edge weights set (set_edge_weights(directional_weights = ...)) and the vertex weight functions set (setDirectionalWeightFunction()).
kappa	Rate parameter of the OU process (kappa > 0).
tau	White-noise scale. The stationary marginal variance is 1/(2*kappa*tau^2) – the same convention as MetricGraph:::r_1(). Note this is NOT reciprocal_tau (used elsewhere in the package as tau = 1/reciprocal_tau); pass tau directly here.
PtE	Evaluation points, a matrix with columns (edge_number, distance_on_edge). Defaults to graph$get_PtE() (the observation locations).
PtE2	Second set of evaluation points for the cross-covariance Cov(u(PtE), u(PtE2)). Defaults to PtE, giving the symmetric n x n covariance matrix at the observation locations.
sigma_source	Anchoring variance at the source vertices (indegree 0). NULL (default) anchors every source at the stationary variance 1/(2*kappa*tau^2), matching Qalpha1_edges_custom(..., stationary_points = "all"). Otherwise a numeric vector giving Var(u) at each source vertex, either named by vertex index (as character) or positional in the order which(graph$get_degrees("indegree") == 0).
normalized	If TRUE (default), distance_on_edge in PtE/PtE2 is in [0, 1]; if FALSE, distances are absolute.
cpp	If TRUE (default), use the C++ numeric setup and the shared C++ covariance fill for oriented in-trees and out-trees. Irregular trees use the generic R fallback. Set to FALSE to use the pure-R implementation throughout.

Details

The covariance between two points s and t is

$$r(s,t) = r(a,a)\, A(a,s)\, A(a,t)$$

where a is the last common ancestor (LCA) of s and t (or covariance 0 if no common ancestor exists), r(a,a) is the marginal variance at a, and

$$A(x,y) = \exp(-\kappa\, d(x,y)) \prod_{v} \beta_v$$

is the transfer factor for the (unique, forward) path from x to y, the product running over the beta_v vertex weights of every vertex crossed strictly between x and y.

In plain terms: the last common ancestor of two points is the furthest-downstream point that is upstream of both of them (it does not exist if the two points lie on disjoint headwater branches, in which case their covariance is exactly zero). The transfer factor A(x,y) is how much of the value at x propagates forward to y: exponential decay over the distance d(x,y), attenuated by the directional weight beta_v at every vertex the path passes through.

This closed form requires graph to represent a directed *tree*: the undirected skeleton must have no cycles, and the directed graph must itself be acyclic. If graph has a directed cycle, is not a tree, or has no directional weight function set, this errors (via the internal directional_ou_setup() validation) – see graph$setDirectionalWeightFunction() to set one.

Value

A dense `matrix` with `nrow(PtE)` rows and `nrow(PtE2)` columns.

See Also

`directional_ou_variance()` for the diagonal only.

Examples

```
edge1 <- rbind(c(0, 0), c(1, 0))
edge2 <- rbind(c(1, 0), c(2, 0))
edge3 <- rbind(c(1, 0), c(1, 1))
graph <- metric_graph$new(edges = list(edge1, edge2, edge3))
graph$set_edge_weights(weights = data.frame(w = c(1, 1, 1)),
                       directional_weights = "w")
graph$setDirectionalWeightFunction(f_in = function(x) sqrt(x / sum(x)))
Sigma <- directional_ou_covariance(graph, kappa = 1, tau = 1,
                                   PtE = rbind(c(1, 0.5), c(2, 0.5)))
Sigma
```

`directional_ou_variance` *Closed-form marginal variance of the directional OU model*

Description

`r(x,x)` at each evaluation point – the diagonal of `directional_ou_covariance()`, computed directly via the marginal variance recursion along each edge, without any last-common-ancestor or transfer-factor work.

Usage

```
directional_ou_variance(
  graph,
  kappa,
  tau,
  PtE = NULL,
  sigma_source = NULL,
  normalized = TRUE,
  cpp = TRUE
)
```

Arguments

`graph`	A `metric_graph` object with directional edge weights set (`set_edge_weights(directional_weights = ...)`) and the vertex weight functions set (`setDirectionalWeightFunction()`).
`kappa`	Rate parameter of the OU process (`kappa > 0`).

tau	White-noise scale. The stationary marginal variance is `1/(2*kappa*tau^2)` – the same convention as `MetricGraph:::r_1()`. Note this is NOT `reciprocal_tau` (used elsewhere in the package as `tau = 1/reciprocal_tau`); pass `tau` directly here.
PtE	Evaluation points, a matrix with columns `(edge_number, distance_on_edge)`. Defaults to `graph$get_PtE()` (the observation locations).
sigma_source	Anchoring variance at the source vertices (indegree 0). `NULL` (default) anchors every source at the stationary variance `1/(2*kappa*tau^2)`, matching `Qalpha1_edges_custom(..., stationary_points = "all")`. Otherwise a numeric vector giving `Var(u)` at each source vertex, either named by vertex index (as character) or positional in the order `which(graph$get_degrees("indegree") == 0)`.
normalized	If `TRUE` (default), `distance_on_edge` in `PtE`/`PtE2` is in `[0, 1]`; if `FALSE`, distances are absolute.
cpp	If `TRUE` (default), use the C++ numeric setup and the shared C++ covariance fill for oriented in-trees and out-trees. Irregular trees use the generic R fallback. Set to `FALSE` to use the pure-R implementation throughout.

Value

A numeric vector of length `nrow(PtE)`.

See Also

`directional_ou_covariance()` for the full closed-form covariance (including the `\deqn{r(s,t) = r(a,a) A(a,s) A(a,t` formula and the tree/acyclicity requirements).

Examples

```
edge1 <- rbind(c(0, 0), c(1, 0))
edge2 <- rbind(c(1, 0), c(2, 0))
edge3 <- rbind(c(1, 0), c(1, 1))
graph <- metric_graph$new(edges = list(edge1, edge2, edge3))
graph$set_edge_weights(weights = data.frame(w = c(1, 1, 1)),
                       directional_weights = "w")
graph$setDirectionalWeightFunction(f_in = function(x) sqrt(x / sum(x)))
directional_ou_variance(graph, kappa = 1, tau = 1,
                        PtE = rbind(c(1, 0.5), c(2, 0.5)))
```

---

drop_na.metric_graph_data

A version of `tidyr::drop_na()` *function for datasets on metric graphs*

Description

Applies `tidyr::drop_na()` function for datasets obtained from a metric graph object.

Usage

```
## S3 method for class 'metric_graph_data'
drop_na(data, ...)
```

Arguments

data	The data list or tidyr::tibble obtained from a metric graph object.
...	Additional parameters to be passed to tidyr::drop_na().

Value

A tidyr::tibble with the resulting selected columns.

---

exp_covariance *Exponential covariance function*

Description

Evaluates the exponential covariance function

$$C(h) = \sigma^2 \exp\{-kappah\}$$

Usage

```
exp_covariance(h, theta)
```

Arguments

h	Distances to evaluate the covariance function at.
theta	A vector c(sigma, kappa), where sigma is the standard deviation and kappa is a range-like parameter.

Value

A vector with the values of the covariance function.

`fetch_osm` *Fetch OpenStreetMap data via the Overpass API*

Description

A drop-in replacement for `osmdata::osmdata_sf()` that sidesteps the Overpass server-status check `osmdata` performs before each download.`fetch_osm()` builds the Overpass query with `osmdata`'s helpers, POSTs it directly with the `curl` package, and then hands the raw response back to `osmdata` for parsing.

Usage

```
fetch_osm(
  bbox,
  key = "highway",
  value = NULL,
  endpoint = "https://overpass-api.de/api/interpreter",
  timeout = 180,
  user_agent = "MetricGraph-R-package",
  cache_path = NULL,
  retries = 2,
  ...
)
```

Arguments

`bbox`	A bounding box, passed to `osmdata::opq()`. May be a length-4 numeric vector `c(xmin, ymin, xmax, ymax)`, an `sf::bbox` object, or any other form accepted by `osmdata::opq()`.
`key`	OSM key to filter on, e.g. `"highway"`. Passed to `osmdata::add_osm_feature()`. Use `NULL` to skip the feature filter and fetch everything in the bbox.
`value`	Optional character vector of values for `key`, e.g. `c("motorway", "motorway_link")`.
`endpoint`	Overpass endpoint URL, or a character vector of URLs that are tried in order. Defaults to the main `overpass-api.de` endpoint. Mirror URLs (e.g. `https://overpass.kumi.systems` work too.
`timeout`	Server-side query timeout in seconds. Forwarded to both `osmdata::opq()` and the HTTP request.
`user_agent`	String sent as the `User-Agent` HTTP header.
`cache_path`	Optional path. If provided and the file already exists, the download is skipped and the cached response is parsed instead. If provided and the file does not exist, the response is saved there for re-use.
`retries`	Number of times a request to each endpoint is retried after a transient failure (HTTP 429 or 5xx, or a connection error), with exponential backoff between attempts.
`...`	Forwarded to `osmdata::add_osm_feature()`.

Value

An osmdata list with elements $osm_points, $osm_lines, $osm_polygons, $osm_multilines, $osm_multipolygons (same structure as osmdata::osmdata_sf()). Tags appear as columns of each sf data frame.

See Also

metric_graph_from_osm() which wraps this and returns a ready-to-use metric_graph.

Examples

```
## Not run:
## All highway ways in a small bbox around San Jose.
osm <- fetch_osm(
  bbox  = c(-122.10, 37.22, -121.80, 37.45),
  key   = "highway",
  value = c("motorway", "motorway_link", "primary", "secondary")
)
head(osm$osm_lines[, c("highway", "oneway", "geometry")])

## End(Not run)
```

---

filter.metric_graph_data

A version of dplyr::filter() *function for datasets on metric graphs*

Description

Applies dplyr::filter() function for datasets obtained from a metric graph object.

Usage

```
## S3 method for class 'metric_graph_data'
filter(.data, ...)
```

Arguments

.data	The data list or tidyr::tibble obtained from a metric graph object.
...	Additional parameters to be passed to dplyr::filter().

Value

A tidyr::tibble with the resulting selected columns.

gg_df.metric_graph_spde_result *Data frame for metric_graph_spde_result objects to be used in 'ggplot2'*

Description

Returns a 'ggplot2'-friendly data-frame with the marginal posterior densities.

Usage

```
## S3 method for class 'metric_graph_spde_result'
gg_df(
  result,
  parameter = result$params,
  transform = TRUE,
  restrict_x_axis = parameter,
 restrict_quantiles = list(sigma = c(0, 1), range = c(0, 1), kappa = c(0, 1), sigma =
    c(0, 1)),
  ...
)
```

Arguments

result	A metric_graph_spde_result object.
parameter	Vector. Which parameters to get the posterior density in the data.frame? The options are sigma, range or kappa.
transform	Should the posterior density be given in the original scale?
restrict_x_axis	Variables to restrict the range of x axis based on quantiles.
restrict_quantiles	List of quantiles to restrict x axis.
...	Not being used.

Value

A data.frame containing the posterior densities.

`glance.graph_lme` *Glance at a* `graph_lme` *object*

Description

Glance accepts a `graph_lme` object and returns a `tidyr::tibble()` with exactly one row of model summaries. The summaries are the square root of the estimated variance of the measurement error, residual degrees of freedom, AIC, BIC, log-likelihood, the type of latent model used in the fit and the total number of observations.

Usage

```
## S3 method for class 'graph_lme'
glance(x, ...)
```

Arguments

`x`	A `graph_lme` object.
`...`	Additional arguments. Currently not used.

Value

A `tidyr::tibble()` with exactly one row and columns:

- `nobs` Number of observations used.
- `sigma` the square root of the estimated residual variance
- `logLik` The log-likelihood of the model.
- `AIC` Akaike's Information Criterion for the model.
- `BIC` Bayesian Information Criterion for the model.
- `deviance` Deviance of the model.
- `df.residual` Residual degrees of freedom.
- `model.type` Type of latent model fitted.

See Also

augment.graph_lme

graph_bru_process_data *Prepare data frames or data lists to be used with 'inlabru' in metric graphs*

Description

Prepare data frames or data lists to be used with 'inlabru' in metric graphs

Usage

```
graph_bru_process_data(
  data,
  edge_number = "edge_number",
  distance_on_edge = "distance_on_edge",
  loc = "loc"
)
```

Arguments

data	A `data.frame` or a `list` containing the covariates, the edge number and the distance on edge for the locations to obtain the prediction.
edge_number	Name of the variable that contains the edge number, the default is `edge_number`.
distance_on_edge	Name of the variable that contains the distance on edge, the default is `distance_on_edge`.
loc	character. Name of the locations to be used in 'inlabru' component.

Value

A list containing the processed data to be used in a user-friendly manner by 'inlabru'.

graph_data_spde *Data extraction for 'spde' models*

Description

Extracts data from metric graphs to be used by 'INLA' and 'inlabru'.

Usage

```
graph_data_spde(
  graph_spde,
  name = "field",
  repl = NULL,
  repl_col = NULL,
  group = NULL,
  group_col = NULL,
  likelihood_col = NULL,
  resp_col = NULL,
  covariates = NULL,
  only_pred = FALSE,
  loc_name = NULL,
  tibble = FALSE,
  drop_na = FALSE,
  drop_all_na = TRUE,
  loc = deprecated()
)
```

Arguments

graph_spde	An inla_metric_graph_spde object built with the graph_spde() function.
name	A character string with the base name of the effect.
repl	Which replicates? If there is no replicates, one can set repl to NULL. If one wants all replicates, then one sets to repl to .all.
repl_col	Column containing the replicates. If the replicate is the internal group variable, set the replicates to ".group". If not replicates, set to NULL.
group	Which groups? If there is no groups, one can set group to NULL. If one wants all groups, then one sets to group to .all.
group_col	Which "column" of the data contains the group variable?
likelihood_col	If only a single likelihood, this variable should be NULL. In case of multiple likelihoods, which column contains the variable indicating the number of the likelihood to be considered?
resp_col	If only a single likelihood, this variable should be NULL. In case of multiple likelihoods, column containing the response variable.
covariates	Vector containing the column names of the covariates. If no covariates, then it should be NULL.
only_pred	Should only return the data.frame to the prediction data?
loc_name	Character with the name of the location variable to be used in 'inlabru' prediction.
tibble	Should the data be returned as a tidyr::tibble?
drop_na	Should the rows with at least one NA for one of the columns be removed? DEFAULT is FALSE. This option is turned to FALSE if only_pred is TRUE.
drop_all_na	Should the rows with all variables being NA be removed? DEFAULT is TRUE. This option is turned to FALSE if only_pred is TRUE.
loc	**[Deprecated]** Use loc_name instead.

Value

An 'INLA' and 'inlabru' friendly list with the data.

`graph_lgcp_sim` *Simulate log-Gaussian Cox processes on metric graphs*

Description

This function simulates point patterns from a log-Gaussian Cox process (LGCP) driven by Whittle-Matérn Gaussian random fields on metric graphs. The intensity function is modeled as $\lambda(s) = exp(\beta + u(s))$, where β is an intercept parameter and $u(s)$ is a Gaussian field with Whittle-Matérn covariance.

Usage

```
graph_lgcp_sim(n = 1, intercept = 0, sigma, range, alpha, graph)
```

Arguments

`n`	Integer. Number of replicate point patterns to simulate. Default is 1.
`intercept`	Numeric scalar or vector. Mean value(s) of the log-intensity field. Can be a constant or vary spatially if provided as a vector of length equal to the number of mesh nodes.
`sigma`	Numeric. Marginal standard deviation parameter of the Gaussian field.
`range`	Numeric. Practical correlation range parameter of the Gaussian field.
`alpha`	Numeric. Smoothness parameter of the Whittle-Matérn field. Currently supports values 1 and 2, corresponding to exponential and Matérn-3/2 covariance functions respectively.
`graph`	A `metric_graph` object with a built mesh. The graph must have an existing mesh (built with `graph$build_mesh()`) and computed FEM matrices (computed with `graph$compute_fem()`).

Details

The function implements a two-step simulation procedure:

1. Simulate the Gaussian random field u(s) using finite element methods
2. Generate point locations using acceptance-rejection sampling from the intensity $\lambda(s) = exp(\beta + u(s))$

The Gaussian field is characterized by the SPDE: $(\kappa^2 - \Delta)^{\alpha/2}\tau u = \mathcal{W}$ where κ, τ are derived from the range and sigma parameters, and $\mathcal{W}$ is white noise.

Value

If n = 1, returns a list with components:

u Numeric vector of the simulated Gaussian field values at mesh nodes

edge_number Integer vector of edge numbers where points were simulated

edge_loc Numeric vector of locations along edges (normalized coordinates)

If n > 1, returns a list of length n, where each element is a list with the above components.

See Also

lgcp_graph for fitting LGCP models, precompute_lgcp_graph for efficient model fitting

Examples

```
## Not run:
# Create a metric graph and build mesh
graph <- metric_graph$new()
graph$build_mesh(h = 0.1)
graph$compute_fem()

# Simulate a single point pattern
lgcp_data <- graph_lgcp_sim(
  intercept = -1,
  sigma = 0.5,
  range = 2,
  alpha = 2,
  graph = graph
)

# Simulate multiple replicates
lgcp_replicates <- graph_lgcp_sim(
  n = 10,
  intercept = -1,
  sigma = 0.5,
  range = 2,
  alpha = 2,
  graph = graph
)

# Plot the simulated intensity
graph$plot_function(X = exp(lgcp_data$u), vertex_size = 0)

## End(Not run)
```

graph_lme *Metric graph linear mixed effects models*

Description

Fitting linear mixed effects model in metric graphs. The random effects can be Gaussian Whittle-Matern fields, discrete Gaussian Markov random fields based on the graph Laplacian, as well as Gaussian random fields with isotropic covariance functions.

Usage

```
graph_lme(
  formula,
  graph,
  model = list(type = "linearModel"),
  which_repl = NULL,
  optim_method = "L-BFGS-B",
  possible_methods = "L-BFGS-B",
  model_options = list(),
  BC = 1,
  previous_fit = NULL,
  fix_coeff = FALSE,
  parallel = FALSE,
  n_cores = parallel::detectCores() - 1,
  optim_controls = list(),
  improve_hessian = FALSE,
  hessian_args = list(),
  check_euclidean = TRUE
)
```

Arguments

formula	Formula object describing the relation between the response variables and the fixed effects.
graph	A metric_graph object.
model	The random effects model that will be used (it also includes the option of not having any random effects). It can be either a character, whose options are 'lm', for linear models without random effects; 'WM1' and 'WM2' for Whittle-Matern models with α=1 and 2, with exact precision matrices, respectively; 'WM' for Whittle-Matern models where one also estimates the smoothness parameter via finite-element method; 'isoExp' for a model with isotropic exponential covariance; 'GL1' and 'GL2' for a SPDE model based on graph Laplacian with α = 1 and 2, respectively. 'WMD1' is the directed Whittle-Matern with α=1. There is also the option to provide it as a list containing the elements type, which can be linearModel, WhittleMatern, graphLaplacian or isoCov. linearModel corresponds to a linear model without random effects.

For WhittleMatern models, that is, if the list contains type = 'WhittleMatern', one can choose between a finite element approximation of the precision matrix by adding fem = TRUE to the list, or to use the exact precision matrix (by setting fem = FALSE). If fem is FALSE, there is also the parameter alpha, to determine the order of the SPDE, which is either 1 or 2. If fem is FALSE and alpha is not specified, then the default value of alpha=1 will be used. If fem is TRUE and one does not specify alpha, it will be estimated from the data. However, if one wants to have alpha fixed to some value, the user can specify either alpha or nu in the list. See the vignettes for examples. Finally, for type 'WhittleMatern', there is an optional argument, rspde_order, that chooses the order of the rational approximation. By default rspde_order is 2. Finally, if one wants to fit a nonstationary model, then fem necessarily needs to be TRUE, and one needs to also supply the matrices B.tau and B.kappa or B.range and B.sigma. For graph-Laplacian models, the list must also contain a parameter alpha (which is 1 by default). For isoCov models, the list must contain a parameter cov_function, containing the covariance function. The function accepts a string input for the following covariance functions: 'exp_covariance', 'WM1', 'WM2', 'GL1', 'GL2'. For another covariance function, the function itself must be provided as the cov_function argument. The default is 'exp_covariance', the exponential covariance. We also have covariance-based versions of the Whittle-Matern and graph Laplacian models, however they are much slower, they are the following (string) values for 'cov_function': 'alpha1' and 'alpha2' for Whittle-Matern fields, and 'GL1' and 'GL2' for graph Laplacian models. Finally, for Whittle-Matern models, there is an additional parameter version, which can be either 1 or 2, to tell which version of the likelihood should be used. Version is 1 by default.

which_repl — Vector or list containing which replicates to consider in the model. If NULL all replicates will be considered.

optim_method — The method to be used with optim function.

possible_methods — Which methods to try in case the optimization fails or the hessian is not positive definite. The options are 'Nelder-Mead', 'L-BFGS-B', 'BFGS', 'CG' and 'SANN'. By default only 'L-BFGS-B' is considered.

model_options — A list containing additional options to be used in the model. Currently, it is possible to fix parameters during the estimation or change the starting values of the parameters. The general structure of the elements of the list is fix_parname and start_parname, where parname stands for the name of the parameter. If fix_parname is not NULL, then the model with be fitted with the parname being fixed at the value that was passed. If start_parname is not NULL, the model will be fitted using the value passed as starting value for parname. the For 'WM' models, the possible elements of the list are: fix_sigma_e, start_sigma_e, fix_nu, start_nu, fix_sigma, start_sigma, fix_range, start_range. Alternatively, one can use fix_sigma_e, start_sigma_e, fix_nu, start_nu, fix_tau, start_tau, fix_kappa, start_kappa. For 'WM1', 'WM2', 'isoExp', 'GL1' and 'GL2' models, the possible elements of the list are fix_sigma_e, start_sigma_e, fix_sigma, start_sigma, fix_range, start_range. Alternatively, one can use fix_sigma_e, start_sigma_e, fix_tau, start_tau, fix_kappa, start_kappa. For 'isoCov' models, the possible values are fix_sigma_e, start_sigma_e, fix_par_vec, start_par_vec. Observe that contrary to the

other models, for 'isoCov' models, both `fix_par_vec` and `start_par_vec` should be given as vectors of the size of the dimension of the vector for the input of the covariance function passed to the 'isoCov' model. Furthermore, for 'isoCov' models, `fix_par_vec` is a logical vector, indicating which parameters to be fixed, and the values will be kept fixed to the values given to `start_par_vec`, one can also use `fix_sigma_e` and `start_sigma_e` for controlling the std. deviation of the measurement error.

`BC` For `WhittleMatern` models, decides which boundary condition to use (0,1). Here, 0 is Neumann boundary conditions and 1 specifies stationary boundary conditions.

`previous_fit` An object of class `graph_lme`. Use the fitted coefficients as starting values.

`fix_coeff` If using a previous fit, should all coefficients be fixed at the starting values?

`parallel` logical. Indicating whether to use `optimParallel()` or not.

`n_cores` Number of cores to be used if parallel is true.

`optim_controls` Additional controls to be passed to `optim()` or `optimParallel()`.

`improve_hessian`
Should a more precise estimate of the hessian be obtained? Turning on might increase the overall time.

`hessian_args` List of controls to be used if `improve_hessian` is `TRUE`. The list can contain the arguments to be passed to the `method.args` argument in the `hessian` function. See the help of the `hessian` function in 'numDeriv' package for details. Observet that it only accepts the "Richardson" method for now, the method "complex" is not supported.

`check_euclidean`
Check if the graph used to compute the resistance distance has Euclidean edges? The graph used to compute the resistance distance has the observation locations as vertices.

Value

A list containing the fitted model.

`graph_spde` *'INLA' implementation of Whittle-Matérn fields for metric graphs*

Description

This function creates an 'INLA' object that can be used in 'INLA' or 'inlabru' to fit Whittle-Matérn fields on metric graphs.

Usage

```
graph_spde(
  graph_object,
  alpha = 1,
  LGCP = FALSE,
  directional = FALSE,
  stationary_endpoints = "all",
  parameterization = c("matern", "spde"),
  start_range = NULL,
  prior_range = NULL,
  range_lower_bound = NULL,
  range_upper_bound = NULL,
  range_prec_inc = 1,
  start_kappa = NULL,
  prior_kappa = NULL,
  kappa_lower_bound = NULL,
  kappa_upper_bound = NULL,
  kappa_prec_inc = 1,
  start_sigma = NULL,
  prior_sigma = NULL,
  sigma_lower_bound = NULL,
  sigma_upper_bound = NULL,
  sigma_prec_inc = 1,
  start_tau = NULL,
  prior_tau = NULL,
  tau_lower_bound = NULL,
  tau_upper_bound = NULL,
  tau_prec_inc = 1,
  factor_start_range = 0.3,
  type_start_range_bbox = "diag",
  shared_lib = "detect",
  debug = FALSE,
  verbose = 0
)
```

Arguments

graph_object	A metric_graph object.
alpha	The order of the SPDE.
LGCP	Logical. If TRUE, the model will be used as a latent model for a Log-Gaussian Cox Process. Default is FALSE.
directional	Should a directional model be used? Currently only implemented for alpha=1.
stationary_endpoints	Which vertices of degree 1 should contain stationary boundary conditions? Set to "all" for all vertices of degree 1, "none" for none of the vertices of degree 1, or pass the indices of the vertices of degree 1 for which stationary conditions are desired.

parameterization
: Which parameterization to be used? The options are 'matern' (sigma and range) and 'spde' (sigma and kappa).

start_range
: Starting value for range parameter.

prior_range
: a list containing the elements meanlog and sdlog, that is, the mean and standard deviation of the range parameter on the log scale. Will not be used if prior.kappa is non-null. If bounds are specified, it can also contain mean and prec for the beta prior.

range_lower_bound
: Lower bound for the range parameter. Default is NULL (no lower bound, or 0).

range_upper_bound
: Upper bound for the range parameter. Default is NULL (no upper bound). If specified, a beta prior will be used.

range_prec_inc
: Amount to increase the precision in the beta prior distribution for the range parameter. Default is 1. Similar to nu.prec.inc in rspde.matern().

start_kappa
: Starting value for kappa.

prior_kappa
: a list containing the elements meanlog and sdlog, that is, the mean and standard deviation of kappa on the log scale. If bounds are specified, it can also contain mean and prec for the beta prior.

kappa_lower_bound
: Lower bound for the kappa parameter. Default is NULL (no lower bound, or 0).

kappa_upper_bound
: Upper bound for the kappa parameter. Default is NULL (no upper bound). If specified, a beta prior will be used.

kappa_prec_inc
: Amount to increase the precision in the beta prior distribution for the kappa parameter. Default is 1.

start_sigma
: Starting value for sigma.

prior_sigma
: a list containing the elements meanlog and sdlog, that is, the mean and standard deviation of sigma on the log scale. If bounds are specified, it can also contain mean and prec for the beta prior.

sigma_lower_bound
: Lower bound for the sigma parameter. Default is NULL (no lower bound, or 0).

sigma_upper_bound
: Upper bound for the sigma parameter. Default is NULL (no upper bound). If specified, a beta prior will be used.

sigma_prec_inc
: Amount to increase the precision in the beta prior distribution for the sigma parameter. Default is 1.

start_tau
: Starting value for tau.

prior_tau
: a list containing the elements meanlog and sdlog, that is, the mean and standard deviation of tau on the log scale. If bounds are specified, it can also contain mean and prec for the beta prior.

tau_lower_bound
: Lower bound for the tau parameter. Default is NULL (no lower bound, or 0).

`tau_upper_bound`
: Upper bound for the tau parameter. Default is NULL (no upper bound). If specified, a beta prior will be used.

`tau_prec_inc`
: Amount to increase the precision in the beta prior distribution for the tau parameter. Default is 1.

`factor_start_range`
: Factor to multiply the max/min dimension of the bounding box to obtain a starting value for range. Default is 0.3.

`type_start_range_bbox`
: Which dimension from the bounding box should be used? The options are 'diag', the default, 'max' and 'min'.

`shared_lib`
: Which shared lib to use for the cgeneric implementation? If "detect", the default, it will use the compiled rSPDE library if one is available locally, and otherwise the models built into the 'INLA' binary. If 'INLA', it will use the models built into the 'INLA' binary. If 'rSPDE', it will use the local installation of the rSPDE package, which requires rSPDE to have been installed from source with its cgeneric library compiled. Otherwise, you can directly supply the path of the .so (or .dll) file.

`debug`
: Should debug be displayed?

`verbose`
: Level of verbosity. 0 is silent, 1 prints basic information, 2 prints more.

Details

This function is used to construct a Matern SPDE model on a metric graph. The latent field u is the solution of the SPDE

$$(\kappa^2 - \Delta)^\alpha u = \sigma W,$$

where W is Gaussian white noise on the metric graph. This model implements exactly the cases in which $\alpha = 1$ or $\alpha = 2$. For a finite element approximation for general α we refer the reader to the 'rSPDE' package and to the Whittle–Matérn fields with general smoothness vignette.

We also have the alternative parameterization $\rho = \frac{\sqrt{8(\alpha-0.5)}}{\kappa}$, which can be interpreted as a range parameter.

Let κ_0 and σ_0 be the starting values for κ and σ, we write $\sigma = \exp\{\theta_1\}$ and $\kappa = \exp\{\theta_2\}$. We assume priors on θ_1 and θ_2 to be normally distributed with mean, respectively, $\log(\sigma_0)$ and $\log(\kappa_0)$, and variance 10. Similarly, if we let ρ_0 be the starting value for ρ, then we write $\rho = \exp\{\theta_2\}$ and assume a normal prior for θ_2, with mean $\log(\rho_0)$ and variance 10.

Value

An 'INLA' object.

graph_spde_basis *Deprecated - Observation/prediction matrices for 'SPDE' models*

Description

Constructs observation/prediction weight matrices for metric graph models.

Usage

```
graph_spde_basis(graph_spde, repl = NULL, drop_na = FALSE, drop_all_na = TRUE)
```

Arguments

graph_spde	An inla_metric_graph_spde object built with the graph_spde() function.
repl	Which replicates? If there is no replicates, or to use all replicates, one can set to NULL.
drop_na	Should the rows with at least one NA for one of the columns be removed? DEFAULT is FALSE.
drop_all_na	Should the rows with all variables being NA be removed? DEFAULT is TRUE.

Value

The observation matrix.

graph_spde_make_A *Deprecated - Observation/prediction matrices for 'SPDE' models*

Description

Constructs observation/prediction weight matrices for metric graph models.

Usage

```
graph_spde_make_A(graph_spde, repl = NULL)
```

Arguments

graph_spde	An inla_metric_graph_spde object built with the graph_spde() function.
repl	Which replicates? If there is no replicates, or to use all replicates, one can set to NULL.

Value

The observation matrix.

graph_starting_values *Starting values for random field models on metric graphs*

Description

Computes appropriate starting values for optimization of Gaussian random field models on metric graphs.

Usage

```
graph_starting_values(
  graph,
  model = c("alpha1", "alpha2", "isoExp", "GL1", "GL2"),
  data = TRUE,
  data_name = NULL,
  range_par = FALSE,
  nu = FALSE,
  manual_data = NULL,
  like_format = FALSE,
  log_scale = FALSE,
  model_options = list(),
  rec_tau = TRUE,
  factor_start_range = 0.3,
  type_start_range_bbox = "diag"
)
```

Arguments

graph	A metric_graph object.
model	Type of model, "alpha1", "alpha2", "isoExp", "GL1", and "GL2" are supported.
data	Should the data be used to obtain improved starting values?
data_name	The name of the response variable in graph$data.
range_par	Should an initial value for range parameter be returned instead of for kappa?
nu	Should an initial value for nu be returned?
manual_data	A vector (or matrix) of response variables.
like_format	Should the starting values be returned with sigma.e as the last element? This is the format for the likelihood constructor from the 'rSPDE' package.
log_scale	Should the initial values be returned in log scale?
model_options	List object containing the model options.
rec_tau	Should a starting value for the reciprocal of tau be given?
factor_start_range	Factor to multiply the max/min/diagonal dimension of the bounding box to obtain a starting value for range. Default is 0.5.
type_start_range_bbox	Which dimension from the bounding box should be used? The options are 'diag', the default, 'max' and 'min'.

Value

A vector, `c(start_sigma_e, start_sigma, start_kappa)`

`lgcp_graph` *Fit log-Gaussian Cox process models on metric graphs*

Description

This function fits log-Gaussian Cox process (LGCP) models for point pattern data on metric graphs using R-INLA. It handles the complex setup required for LGCP modeling, including creation of integration points, data preparation, and interface with INLA's Poisson likelihood framework. The function supports both exact and rational SPDE approximations, multiple replicates, and efficient refitting using precomputed quantities.

Usage

```
lgcp_graph(
  formula,
  graph,
  interpolate = TRUE,
  manual_integration_points = NULL,
  manual_covariates = NULL,
  use_current_mesh = TRUE,
  new_h = NULL,
  new_n = NULL,
  repl = ".all",
  repl_col = ".group",
  clone_graph = TRUE,
  precomputed_data = NULL,
  ...
)
```

Arguments

`formula`	A formula object specifying the model structure. Should follow INLA syntax, e.g., `y ~ covariate + f(field, model = spde_model)` where `field` is the spatial random effect and `spde_model` is an SPDE model object.
`graph`	A `metric_graph` object containing the network structure and point pattern data. Must have observations added via `add_observations()`.
`interpolate`	Logical. If `TRUE` (default), interpolate covariate values from graph data to integration points. If `FALSE`, use `manual_covariates`.
`manual_integration_points`	Data frame with columns `edge_number`, `distance_on_edge`, and `E` (integration weights). If `NULL`, automatic integration points are created.

`manual_covariates`
: Data frame containing covariate values at integration points when `interpolate = FALSE`. Must include a `.group` column for replicates if using replicated data.

`use_current_mesh`
: Logical. If `TRUE` (default), use the existing mesh in the graph as integration points. If `FALSE`, create a new mesh.

`new_h`
: Numeric. Mesh resolution for creating a new mesh when `use_current_mesh = FALSE`. Smaller values create finer meshes.

`new_n`
: Integer. Alternative to `new_h`, specifies the approximate number of mesh nodes for the new mesh.

`repl`
: Character vector or `".all"`. Specifies which replicates to include in the model. Use `".all"` to include all available replicates.

`repl_col`
: Character. Name of the column in the graph data that contains replicate identifiers. Default is `".group"`.

`clone_graph`
: Logical. If `TRUE` (default), clone the graph to avoid modifying the original object. If `FALSE`, work directly on the original graph (faster but modifies the input). Only used when `precomputed_data` is `NULL`.

`precomputed_data`
: Optional object of class `"precomputed_lgcp"` from `precompute_lgcp_graph()`. Enables efficient refitting with different formulas using the same spatial structure and covariates.

`...`
: Additional arguments passed to `INLA::inla()`, such as `control.inla`, `control.predictor`, `control.compute`, etc.

Details

The function implements LGCP modeling using the approach of Simpson et al. (2016), where the log-Gaussian Cox process with intensity $\lambda(s) = exp(\eta(s))$ is approximated using a Poisson likelihood with carefully constructed integration points and weights.

The key steps are:

1. Create integration points (typically mesh nodes) across the graph
2. Set up data with observed points (response = 1, weights = 0) and integration points (response = 0, weights = integration weights)
3. Fit using Poisson regression with the constructed weights as exposure

The spatial component can be modeled using:

- Exact SPDE models via `graph_spde()` (slower setup, exact likelihood)
- Rational SPDE approximations via `rspde.metric_graph()` (faster, approximate)

Value

An object of class `"inla"` containing the fitted LGCP model. This includes posterior marginal distributions for model parameters, fitted values, and other standard INLA output components. Use `spde_metric_graph_result()` to extract spatial parameter estimates in their original scale.

Performance

For fitting multiple models with the same spatial structure:

- Use `precompute_lgcp_graph()` first, then `lgcp_graph()` with `precomputed_data` for substantial speedups
- Set `clone_graph = FALSE` for additional performance gains when you don't need to preserve the original graph

References

Simpson, D., Illian, J., Lindgren, F., Sørbye, S., & Rue, H. (2016). Going off grid: Computationally efficient inference for log-Gaussian Cox processes. Biometrika, 103(1), 49-70.

See Also

`graph_lgcp_sim` for simulating LGCP data, `precompute_lgcp_graph` for efficient model refitting, `graph_spde` for exact SPDE models, `spde_metric_graph_result` for extracting spatial parameter estimates

Examples

```
## Not run:
# Setup: Create graph with mesh and add point pattern data
graph <- metric_graph$new()
graph$build_mesh(h = 0.1)
graph$add_observations(data = your_point_data,
                       edge_number = "edge_id",
                       distance_on_edge = "location")

# Create SPDE model
spde_model <- graph_spde(graph, alpha = 1)
# or: rspde_model <- rspde.metric_graph(graph, nu = 1.5)

# Fit basic LGCP model
fit1 <- lgcp_graph(y ~ 1 + f(field, model = spde_model),
                   graph = graph)

# Fit model with covariates
fit2 <- lgcp_graph(y ~ elevation + temperature +
                       f(field, model = spde_model),
                   graph = graph)

# Extract spatial parameter estimates
spde_result <- spde_metric_graph_result(fit2, "field", spde_model)
summary(spde_result)

# Efficient fitting of multiple models
precomputed <- precompute_lgcp_graph(
  graph = graph,
  resp_variable_name = "y",
  model_name = "field",
```

```
  spde_model = spde_model,
  covariates = c("elevation", "temperature", "slope")
)

# Now fit multiple models efficiently
fit_a <- lgcp_graph(y ~ elevation + f(field, model = spde_model),
                    graph = graph, precomputed_data = precomputed)
fit_b <- lgcp_graph(y ~ elevation + temperature + f(field, model = spde_model),
                    graph = graph, precomputed_data = precomputed)
fit_c <- lgcp_graph(y ~ slope + f(field, model = spde_model),
                    graph = graph, precomputed_data = precomputed)

# Model with replicates
fit_rep <- lgcp_graph(y ~ covariate + f(field, model = spde_model,
                                        replicate = field.repl),
                      graph = graph)

## End(Not run)
```

`linnet.to.graph` *Convert a* `linnet` *object to a metric graph object*

Description

This function converts a `linnet` object (from the `spatstat` package) into a metric graph object.

Usage

```
linnet.to.graph(linnet.object, crs, ...)
```

Arguments

`linnet.object`	A `linnet` object to be converted.
`crs`	The coordinate reference system of the graph.
`...`	Additional arguments to be passed to the `metric_graph` constructor.

Value

A metric graph object with edges defined by the network.

logo_lines *Create lines for package name*

Description

Create lines for package name

Usage

```
logo_lines()
```

Value

`SpatialLines` object with package name.

make_Q_euler *Space-time precision operator Euler discretization*

Description

The precision matrix for all vertices for space-time field

Usage

```
make_Q_euler(graph, t, kappa, rho, gamma, alpha, beta, sigma, theta = 1)
```

Arguments

`graph`	A `metric_graph` object.
`t`	Vector of time points.
`kappa`	Spatial range parameter.
`rho`	Drift parameter.
`gamma`	Temporal range parameter.
`alpha`	Smoothness parameter (integer) for spatial operator.
`beta`	Smoothness parameter (integer) for Q-Wiener process.
`sigma`	Variance parameter.
`theta`	Parameter theta for the Euler scheme.

Value

Precision matrix.

make_Q_spacetime *Space-time precision operator discretization*

Description

The precision matrix for all vertices for space-time field.

Usage

```
make_Q_spacetime(graph, t, kappa, rho, gamma, alpha, beta, sigma)
```

Arguments

graph	A metric_graph object.
t	Vector of time points.
kappa	Spatial range parameter.
rho	Drift parameter.
gamma	Temporal range parameter.
alpha	Smoothness parameter (integer) for spatial operator.
beta	Smoothness parameter (integer) for Q-Wiener process.
sigma	Variance parameter.

Value

Precision matrix.

match_mesh_data *Match Data Frame Rows to Graph Mesh Order*

Description

Reorders the rows of a data frame to align with the specific ordering of points along the edges of a metric graph's mesh. This ensures that data associated with locations on the graph are correctly aligned with the graph's internal mesh structure, which is essential for spatial operations and visualizations.

Usage

```
match_mesh_data(
  graph,
  data,
  edge_col = ".edge_number",
  dist_col = ".distance_on_edge"
)
```

Arguments

graph	A metric graph object with a mesh component containing VtE.
data	A data.frame to be reordered. It must contain columns for edge numbers and distances along those edges.
edge_col	Character. Name of the column in data that contains the edge numbers. Default: ".edge_number".
dist_col	Character. Name of the column in data that contains the distance along the edge for each point. Default: ".distance_on_edge".

Value

A reordered data.frame where rows correspond to the order of points in graph$mesh$VtE. If the input data was an sf object, the returned object will also be an sf object with its geometry reordered accordingly.

---

metric_graph *Metric graph*

Description

Class representing a general metric graph.

Details

A graph object created from vertex and edge matrices, or from an sp::SpatialLines object where each line is representing and edge. For more details, see the vignette: vignette("metric_graph", package = "MetricGraph")

Value

Object of R6Class for creating metric graphs.

Public fields

V Matrix with positions in Euclidean space of the vertices of the graph.

nV The number of vertices.

E Matrix with the edges of the graph, where each row represents an edge, E[i,1] is the vertex at the start of the ith edge and E[i,2] is the vertex at the end of the edge.

nE The number of edges.

edge_lengths Vector with the lengths of the edges in the graph.

C Constraint matrix used to set Kirchhoff constraints.

CoB Change-of-basis object used for Kirchhoff constraints.

PtV Vector with the indices of the vertices which are observation locations.

mesh Mesh object used for plotting.

`edges` The coordinates of the edges in the graph.

`DirectionalWeightFunction_in` Function for inwards weights in directional models

`DirectionalWeightFunction_out` Function for outwards weights in directional models

`vertices` The coordinates of the vertices in the graph, along with several attributes.

`geo_dist` Geodesic distances between the vertices in the graph.

`res_dist` Resistance distances between the observation locations.

`Laplacian` The weighted graph Laplacian of the vertices in the graph. The weights are given by the edge lengths.

`characteristics` List with various characteristics of the graph.

Methods

Public methods:

- `metric_graph$new()`
- `metric_graph$remove_small_circles()`
- `metric_graph$get_edges()`
- `metric_graph$get_bounding_box()`
- `metric_graph$get_vertices()`
- `metric_graph$export()`
- `metric_graph$leaflet()`
- `metric_graph$mapview()`
- `metric_graph$set_edge_weights()`
- `metric_graph$get_edge_weights()`
- `metric_graph$get_vertices_incomp_dir()`
- `metric_graph$summary()`
- `metric_graph$print()`
- `metric_graph$compute_characteristics()`
- `metric_graph$check_euclidean()`
- `metric_graph$check_distance_consistency()`
- `metric_graph$compute_geodist()`
- `metric_graph$compute_geodist_PtE()`
- `metric_graph$compute_geodist_mesh()`
- `metric_graph$compute_resdist()`
- `metric_graph$compute_resdist_PtE()`
- `metric_graph$get_degrees()`
- `metric_graph$compute_PtE_edges()`
- `metric_graph$compute_resdist_mesh()`
- `metric_graph$compute_laplacian()`
- `metric_graph$prune_vertices()`
- `metric_graph$set_manual_edge_lengths()`
- `metric_graph$get_groups()`
- `metric_graph$get_PtE()`

- metric_graph$get_edge_lengths()
- metric_graph$get_locations()
- metric_graph$observation_to_vertex()
- metric_graph$edgeweight_to_data()
- metric_graph$get_mesh_locations()
- metric_graph$clear_observations()
- metric_graph$process_data()
- metric_graph$add_observations()
- metric_graph$mutate_weights()
- metric_graph$select_weights()
- metric_graph$filter_weights()
- metric_graph$summarise_weights()
- metric_graph$drop_na_weights()
- metric_graph$mutate()
- metric_graph$drop_na()
- metric_graph$select()
- metric_graph$filter()
- metric_graph$summarise()
- metric_graph$get_data()
- metric_graph$setDirectionalWeightFunction()
- metric_graph$buildDirectionalConstraints()
- metric_graph$compute_directional_covariance()
- metric_graph$buildC()
- metric_graph$build_mesh()
- metric_graph$get_version()
- metric_graph$is_disconnected()
- metric_graph$get_components()
- metric_graph$get_largest()
- metric_graph$which_component()
- metric_graph$compute_fem()
- metric_graph$compute_mesh_weights()
- metric_graph$mesh_A()
- metric_graph$fem_basis()
- metric_graph$VtEfirst()
- metric_graph$plot()
- metric_graph$plot_connections()
- metric_graph$is_tree()
- metric_graph$plot_function()
- metric_graph$plot_movie()
- metric_graph$add_mesh_observations()
- metric_graph$get_initial_graph()
- metric_graph$update_graph()

- metric_graph$coordinates()
- metric_graph$clone()

`metric_graph$new()`: Create a new `metric_graph` object.

Usage:

```
metric_graph$new(
  edges = NULL,
  V = NULL,
  E = NULL,
  vertex_unit = NULL,
  length_unit = NULL,
  edge_weights = NULL,
  kirchhoff_weights = NULL,
  directional_weights = NULL,
  longlat = NULL,
  crs = NULL,
  proj4string = NULL,
  which_longlat = "sp",
  include_obs = NULL,
  include_edge_weights = NULL,
  project = FALSE,
  project_data = FALSE,
  which_projection = "Winkel tripel",
  manual_edge_lengths = NULL,
  perform_merges = NULL,
  approx_edge_PtE = TRUE,
  tolerance = list(vertex_vertex = 0.001, vertex_edge = 0.001, edge_edge = 0),
  check_connected = TRUE,
  remove_deg2 = FALSE,
  merge_close_vertices = NULL,
  factor_merge_close_vertices = 1,
  remove_circles = FALSE,
  auto_remove_point_edges = TRUE,
  verbose = 1,
  add_obs_options = list(return_removed = FALSE, verbose = verbose),
  lines = deprecated(),
  .assemble = NULL
)
```

Arguments:

`edges` A list containing coordinates as `m x 2` matrices (that is, of `matrix` type) or m x 2 data frames (`data.frame` type) of sequence of points connected by straightlines. Alternatively, you can also prove an object of type `SSN`, `osmdata_sp`, `osmdata_sf`, `SpatialLinesDataFrame` or `SpatialLines` (from `sp` package) or `MULTILINESTRING` (from `sf` package).

`V` n x 2 matrix with Euclidean coordinates of the n vertices. If non-NULL, no merges will be performed.

`E` m x 2 matrix where each row represents one of the m edges. If non-NULL, no merges will be performed.

vertex_unit The unit in which the vertices are specified. The options are 'degree' (the great circle distance in km), 'km', 'm' and 'miles'. The default is NULL, which means no unit. However, if you set length_unit, you need to set vertex_unit.

length_unit The unit in which the lengths will be computed. The options are 'km', 'm' and 'miles'. The default, when longlat is TRUE, or an sf or sp objects are provided, is 'km'.

edge_weights Either a number, a numerical vector with length given by the number of edges, providing the edge weights, or a data.frame with the number of rows being equal to the number of edges, where each row gives a vector of weights to its corresponding edge. Can be changed by using the set_edge_weights() method.

kirchhoff_weights If non-null, the name (or number) of the column of edge_weights that contain the Kirchhoff weights. Must be equal to 1 (or TRUE) in case edge_weights is a single number and those are the Kirchhoff weights.

directional_weights If non-null, the name (or number) of the column of edge_weights that contain the directional weights. The default is the first column of the edge weights.

longlat There are three options: NULL, TRUE or FALSE. If NULL (the default option), the edges argument will be checked to see if there is a CRS or proj4string available, if so, longlat will be set to TRUE, otherwise, it will be set to FALSE. If TRUE, then it is assumed that the coordinates are given. in Longitude/Latitude and that distances should be computed in meters. If TRUE it takes precedence over vertex_unit and length_unit, and is equivalent to vertex_unit = 'degree' and length_unit = 'm'.

crs Coordinate reference system to be used in case longlat is set to TRUE and which_longlat is sf. Object of class crs. The default choice, if the edges object does not have CRS nor proj4string, is sf::st_crs(4326).

proj4string Projection string of class CRS-class to be used in case longlat is set to TRUE and which_longlat is sp. The default choice, if the edges object does not have CRS nor proj4string, is sp::CRS("+proj=longlat +datum=WGS84").

which_longlat Compute the distance using which package? The options are sp and sf. The default is sp.

include_obs If the object is of class SSN, should the observations be added? If NULL and the edges are of class SSN, the data will be automatically added. If FALSE, the data will not be added. Alternatively, one can set this argument to the numbers or names of the columns of the observations to be added as observations.

include_edge_weights If the object is of class SSN, osmdata_sp, osmdata_sf, SpatialLinesDataFrame, MULTILINESTRING, LINESTRING, sfc_LINESTRING, sfc_MULTILINESTRING, should the edge data (if any) be added as edge weights? If NULL, the edge data will be added as edge weights, if FALSE they will not be added. Alternatively, one can set this argument to the numbers or names of the columns of the edge data to be added as edge weights.

project If longlat is TRUE should a projection be used to compute the distances to be used for the tolerances (see tolerance below)? The default is FALSE. When TRUE, the construction of the graph is faster.

project_data If longlat is TRUE should the vertices be project to planar coordinates? The default is FALSE. When TRUE, the construction of the graph is faster.

which_projection Which projection should be used in case project is TRUE? The options are Robinson, Winkel tripel or a proj4string. The default is Winkel tripel.

manual_edge_lengths If non-NULL, a vector containing the edges lengths, and all the quantities related to edge lengths will be computed in terms of these. If merges are performed, it is likely that the merges will override the manual edge lengths. In such a case, to provide

manual edge lengths, one should either set the perform_merges argument to FALSE or use the set_manual_edge_lengths() method.

perform_merges There are three options, NULL, TRUE or FALSE. The default option is NULL. If NULL, it will be set to FALSE unless 'edges', 'V' and 'E' are NULL, in which case it will be set to TRUE. If FALSE, this will take priority over the other arguments, and no merges (except the optional merge_close_vertices below) will be performed. Note that the merge on the additional merge_close_vertices might still be performed, if it is set to TRUE.

approx_edge_PtE Should the relative positions on the edges be approximated? The default is TRUE. If FALSE, the speed can be considerably slower, especially for large metric graphs.

tolerance List that provides tolerances during the construction of the graph:

- vertex_vertex Vertices that are closer than this number are merged (default = 1e-7).
- vertex_edge If a vertex at the end of one edge is closer than this number to another edge, this vertex is connected to that edge (default = 1e-7). Previously vertex_line, which is now deprecated.
- edge_edge If two edges at some point are closer than this number, a new vertex is added at that point and the two edges are connected (default = 0).
- vertex_line, Deprecated. Use vertex_edge instead.
- line_line, Deprecated. Use edge_edge instead.

In case longlat = TRUE, the tolerances are given in length_unit.

check_connected If TRUE, it is checked whether the graph is connected and a warning is given if this is not the case.

remove_deg2 Set to TRUE to remove all vertices of degree 2 in the initialization. Default is FALSE.

merge_close_vertices Should an additional step to merge close vertices be done? The options are NULL (the default), TRUE or FALSE. If NULL, it will be determined automatically. If TRUE this step will be performed even if perfom_merges is set to FALSE.

factor_merge_close_vertices Which factor to be multiplied by tolerance vertex_vertex when merging close vertices at the additional step?

remove_circles All circlular edges with a length smaller than this number are removed. If TRUE, the vertex_vertex tolerance will be used. If FALSE, no circles will be removed.

auto_remove_point_edges Should edges of length zero, that is, edges that are actually points, be automatically removed?

verbose Print progress of graph creation. There are 3 levels of verbose, level 0, 1 and 2. In level 0, no messages are printed. In level 1, only messages regarding important steps are printed. Finally, in level 2, messages detailing all the steps are printed. The default is 1.

add_obs_options List containing additional options to be passed to the add_observations() method when adding observations from SSN data?

lines **[Deprecated]** Use edges instead.

.assemble Expert option for assembling the graph by skipping the entire constructor pipeline and instead populate the R6 object directly from a pre-computed list. Intended for internal use.

Details: A graph object can be initialized in two ways. The first method is to specify V and E. In this case, all edges are assumed to be straight lines. The second option is to specify the graph via the lines input. In this case, the vertices are set by the end points of the lines. Thus, if two lines are intersecting somewhere else, this will not be viewed as a vertex.

Returns: A metric_graph object.

`metric_graph$remove_small_circles()`: Sets the edge weights

Usage:

```
metric_graph$remove_small_circles(tolerance, verbose = 1)
```

Arguments:

`tolerance` Tolerance at which circles with length less than this will be removed.

`verbose` Print progress of graph creation. There are 3 levels of verbose, level 0, 1 and 2. In level 0, no messages are printed. In level 1, only messages regarding important steps are printed. Finally, in level 2, messages detailing all the steps are printed. The default is 1.

Returns: No return value. Called for its side effects.

`metric_graph$get_edges()`: Exports the edges of the MetricGraph object as an `sf` or `sp`.

Usage:

```
metric_graph$get_edges(format = c("sf", "sp", "list"))
```

Arguments:

`format` The format for the exported object. The options are `sf` (default), `sp` and `list`.

Returns: For `format == "sf"`, the function returns an `sf` object of `LINESTRING` geometries, where the associated data frame includes edge weights.
For `format == "sp"`, the function returns a `SpatialLinesDataFrame` where the data frame includes edge weights.

`metric_graph$get_bounding_box()`: Bounding box of the metric graph

Usage:

```
metric_graph$get_bounding_box(format = "sf")
```

Arguments:

`format` If the metric graph has a coordinate reference system, the format for the exported object. The options are `sf` (default), `sp` and `matrix`.

Returns: A bounding box of the metric graph

`metric_graph$get_vertices()`: Exports the vertices of the MetricGraph object as an `sf`, `sp` or as a matrix.

Usage:

```
metric_graph$get_vertices(format = c("sf", "sp", "list"))
```

Arguments:

`format` The format for the exported object. The options are `sf` (default), `sp` and `matrix`.

Returns: For `which_format == "sf"`, the function returns an `sf` object of `POINT` geometries.
For `which_format == "sp"`, the function returns a `SpatialPointsDataFrame` object.

`metric_graph$export()`: Exports the MetricGraph object as an `sf` or `sp` object.

Usage:

```
metric_graph$export(format = "sf")
```

Arguments:

`format` The format for the exported object. The options are `sf` (default) and `sp`.

Returns: Returns a list with three elements: `edges`, `vertices`, and `data`.
For `format == "sf"`, `edges` is an `sf` object of `LINESTRING` geometries with edge weights, and `vertices` and `data` are `sf` objects with `POINT` geometries.
For `format == "sp"`, `edges` is a `SpatialLinesDataFrame` with edge weights, and `vertices` and `data` are `SpatialPointsDataFrame`.

`metric_graph$leaflet()`**:** Return the metric graph as a `leaflet::leaflet()` object to be built upon.

Usage:

```
metric_graph$leaflet(
  width = NULL,
  height = NULL,
  padding = 0,
  options = leafletOptions(),
  elementId = NULL,
  sizingPolicy = leafletSizingPolicy(padding = padding)
)
```

Arguments:

`width` the width of the map

`height` the height of the map

`padding` the padding of the map

`options` the map options

`elementId` Use an explicit element ID for the widget (rather than an automatically generated one).

`sizingPolicy` htmlwidgets sizing policy object. Defaults to `leafletSizingPolicy()`.

`metric_graph$mapview()`**:** Returns a `mapview::mapview()` object of the metric graph

Usage:

```
metric_graph$mapview(...)
```

Arguments:

`...` Additional arguments to be passed to `mapview::mapview()`. The `x` argument of mapview, containing the metric graph is already passed internally.

`metric_graph$set_edge_weights()`**:** Sets the edge weights

Usage:

```
metric_graph$set_edge_weights(
  weights = NULL,
  kirchhoff_weights = NULL,
  directional_weights = NULL,
  verbose = 0
)
```

Arguments:

`weights` Either a number, a numerical vector with length given by the number of edges, providing the edge weights, or a `data.frame` with the number of rows being equal to the number of edges, where each row gives a vector of weights to its corresponding edge.

`kirchhoff_weights` If non-null, the name (or number) of the column of `weights` that contain the Kirchhoff weights. Must be equal to 1 (or `TRUE`) in case `weights` is a single number and those are the Kirchhoff weights.

`directional_weights` If non-null, the name (or number) of the column of `weights` that contain the directional weights.

`verbose` There are 3 levels of verbose, level 0, 1 and 2. In level 0, no messages are printed. In level 1, only messages regarding important steps are printed. Finally, in level 2, messages detailing all the steps are printed. The default is 1.

Returns: No return value. Called for its side effects.

`metric_graph$get_edge_weights()`**:** Gets the edge weights

Usage:

```
metric_graph$get_edge_weights(
  data.frame = FALSE,
  format = c("tibble", "sf", "sp", "list"),
  tibble = deprecated()
)
```

Arguments:

`data.frame` If the edge weights are given as vectors, should the result be returned as a data.frame?

`format` Which format should the data be returned? The options are `tibble` for `tidyr::tibble`, `sf` for `POINT`, `sp` for `SpatialPointsDataFrame` and `list` for the internal list format.

`tibble` **[Deprecated]** Use `format` instead.

Returns: A vector or `data.frame` containing the edge weights.

`metric_graph$get_vertices_incomp_dir()`**:** Gets vertices with incompatible directions

Usage:

```
metric_graph$get_vertices_incomp_dir()
```

Returns: A vector containing the vertices with incompatible directions.

`metric_graph$summary()`**:** Prints a summary of various informations of the graph

Usage:

```
metric_graph$summary(
  messages = FALSE,
  compute_characteristics = NULL,
  check_euclidean = NULL,
  check_distance_consistency = NULL
)
```

Arguments:

`messages` Should message explaining how to build the results be given for missing quantities?

`compute_characteristics` Should the characteristics of the graph be computed? If `NULL` it will be determined based on the size of the graph.

check_euclidean Check if the graph has Euclidean edges? If NULL it will be determined based on the size of the graph.

check_distance_consistency Check the distance consistency assumption? If NULL it will be determined based on the size of the graph.

Returns: No return value. Called for its side effects.

`metric_graph$print()`: Prints various characteristics of the graph

Usage:

```
metric_graph$print()
```

Returns: No return value. Called for its side effects.

`metric_graph$compute_characteristics()`: Computes various characteristics of the graph

Usage:

```
metric_graph$compute_characteristics(check_euclidean = FALSE)
```

Arguments:

check_euclidean Also check if the graph has Euclidean edges? This essentially means that the distance consistency check will also be perfomed. If the graph does not have Euclidean edges due to another reason rather than the distance consistency, then it will already be indicated that the graph does not have Euclidean edges.

Returns: No return value. Called for its side effects. The computed characteristics are stored in the characteristics element of the metric_graph object.

`metric_graph$check_euclidean()`: Check if the graph has Euclidean edges.

Usage:

```
metric_graph$check_euclidean()
```

Returns: Returns TRUE if the graph has Euclidean edges, or FALSE otherwise. The result is stored in the characteristics element of the metric_graph object. The result is displayed when the graph is printed.

`metric_graph$check_distance_consistency()`: Checks distance consistency of the graph.

Usage:

```
metric_graph$check_distance_consistency()
```

Returns: No return value. The result is stored in the characteristics element of the metric_graph object. The result is displayed when the graph is printed.

`metric_graph$compute_geodist()`: Computes shortest path distances between the vertices in the graph

Usage:

```
metric_graph$compute_geodist(
  obs = TRUE,
  include_vertices = FALSE,
  all_groups = FALSE,
  group = NULL,
  verbose = 0,
  full = lifecycle::deprecated()
)
```

Arguments:

`obs` Should the geodesic distances be computed at the observation locations or only at vertices?

`include_vertices` If `obs` is `TRUE`, should the vertex locations be included in the resulting distance matrix?

`all_groups` Should the geodesic distances be computed for all the available locations accross all groups? If `FALSE`, it will be computed separately for the locations of each group.

`group` Vector or list containing which groups to compute the distance for. If `NULL`, it will be computed for all groups.

`verbose` Print progress of the computation of the geodesic distances. There are 3 levels of verbose, level 0, 1 and 2. In level 0, no messages are printed. In level 1, only messages regarding important steps are printed. Finally, in level 2, messages detailing all the steps are printed. The default is 1.

`full` **[Deprecated]** Use `all_groups` instead.

Returns: No return value. Called for its side effects. The computed geodesic distances are stored in the `geo_dist` element of the `metric_graph` object.

`metric_graph$compute_geodist_PtE()`**:** Computes shortest path distances among points on graph edges, preserving the input order and duplicate points.

Usage:

```
metric_graph$compute_geodist_PtE(
  PtE,
  normalized = TRUE,
  include_vertices = TRUE,
  verbose = 0
)
```

Arguments:

`PtE` Two-column matrix of edge numbers and positions on edges.

`normalized` Are edge positions fractions of edge length?

`include_vertices` Should the original vertices be included in the distance matrix?

`verbose` Print progress of the computation of the geodesic distances. There are 3 levels of verbose, level 0, 1 and 2. In level 0, no messages are printed. In level 1, only messages regarding important steps are printed. Finally, in level 2, messages detailing all the steps are printed. The default is 0.

Returns: A distance matrix with one row and column per input point, plus one per graph vertex at the beginning if `include_vertices` is TRUE.

`metric_graph$compute_geodist_mesh()`**:** Computes shortest path distances between the vertices in the mesh.

Usage:

```
metric_graph$compute_geodist_mesh()
```

Returns: No return value. Called for its side effects. The geodesic distances on the mesh are stored in `mesh$geo_dist` in the `metric_graph` object.

`metric_graph$compute_resdist()`**:** Computes the resistance distance between the observation locations.

Usage:

```
metric_graph$compute_resdist(
  full = FALSE,
  obs = TRUE,
  group = NULL,
  check_euclidean = FALSE,
  include_vertices = FALSE,
  verbose = 0
)
```

Arguments:

`full` Should the resistance distances be computed for all the available locations. If `FALSE`, it will be computed separately for the locations of each group.

`obs` Should the resistance distances be computed at the observation locations?

`group` Vector or list containing which groups to compute the distance for. If `NULL`, it will be computed for all groups.

`check_euclidean` Check if the graph used to compute the resistance distance has Euclidean edges? The graph used to compute the resistance distance has the observation locations as vertices.

`include_vertices` Should the vertices of the graph be also included in the resulting matrix when using `FULL=TRUE`?

`verbose` Print progress of the computation of the resistance distances. There are 3 levels of verbose, level 0, 1 and 2. In level 0, no messages are printed. In level 1, only messages regarding important steps are printed. Finally, in level 2, messages detailing all the steps are printed. The default is 1.

Returns: No return value. Called for its side effects. The geodesic distances are stored in the `res_dist` element of the `metric_graph` object.

`metric_graph$compute_resdist_PtE()`**:** Computes the resistance distance between the observation locations.

Usage:

```
metric_graph$compute_resdist_PtE(
  PtE,
  normalized = TRUE,
  include_vertices = FALSE,
  check_euclidean = FALSE,
  verbose = 0
)
```

Arguments:

`PtE` Points to compute the metric for.

`normalized` Are the locations in PtE in normalized distance?

`include_vertices` Should the original vertices be included in the Laplacian matrix?

`check_euclidean` Check if the graph used to compute the resistance distance has Euclidean edges? The graph used to compute the resistance distance has the observation locations as vertices.

verbose Print progress of the computation of the resistance distances. There are 3 levels of verbose, level 0, 1 and 2. In level 0, no messages are printed. In level 1, only messages regarding important steps are printed. Finally, in level 2, messages detailing all the steps are printed. The default is 1.

Returns: A matrix containing the resistance distances.

`metric_graph$get_degrees()`: Returns the degrees of the vertices in the metric graph.

Usage:

```
metric_graph$get_degrees(which = "degree")
```

Arguments:

which If "degree", returns the degree of the vertex. If "indegree", returns the indegree, and if "outdegree", it returns the outdegree.

Returns: A vector containing the degrees of the vertices.

`metric_graph$compute_PtE_edges()`: Computes the relative positions of the coordinates of the edges and save it as an attribute to each edge. This improves the quality of plots obtained by the `plot_function()` method, however it might be costly to compute.

Usage:

```
metric_graph$compute_PtE_edges(approx = TRUE, verbose = 0)
```

Arguments:

approx Should the computation of the relative positions be approximate? Default is TRUE. If FALSE, the speed can be considerably slower, especially for large metric graphs.

verbose Level of verbosity, 0, 1 or 2. The default is 0.

Returns: No return value, called for its side effects.

`metric_graph$compute_resdist_mesh()`: Computes the resistance metric between the vertices in the mesh.

Usage:

```
metric_graph$compute_resdist_mesh()
```

Returns: No return value. Called for its side effects. The geodesic distances on the mesh are stored in the `mesh$res_dist` element in the `metric_graph` object.

`metric_graph$compute_laplacian()`: Computes the weigthed graph Laplacian for the graph.

Usage:

```
metric_graph$compute_laplacian(
  full = FALSE,
  obs = TRUE,
  group = NULL,
  verbose = 0
)
```

Arguments:

full Should the resistance distances be computed for all the available locations. If FALSE, it will be computed separately for the locations of each group.

obs Should the resistance distances be computed at the observation locations? It will only compute for locations in which there is at least one observations that is not NA.

group Vector or list containing which groups to compute the Laplacian for. If NULL, it will be computed for all groups.

verbose Print progress of the computation of the Laplacian. There are 3 levels of verbose, level 0, 1 and 2. In level 0, no messages are printed. In level 1, only messages regarding important steps are printed. Finally, in level 2, messages detailing all the steps are printed. The default is 1.

Returns: No reutrn value. Called for its side effects. The Laplacian is stored in the Laplacian element in the metric_graph object.

`metric_graph$prune_vertices()`**:** Removes vertices of degree 2 from the metric graph.

Usage:

```
metric_graph$prune_vertices(
  check_weights = TRUE,
  check_circles = TRUE,
  verbose = FALSE
)
```

Arguments:

check_weights If TRUE will only prune edges with different weights.

check_circles If TRUE will not prune a vertex such that the resulting edge is a circle.

verbose Print progress of pruning. There are 3 levels of verbose, level 0, 1 and 2. In level 0, no messages are printed. In level 1, only messages regarding important steps are printed. Finally, in level 2, messages detailing all the steps are printed. The default is 1.

Details: Vertices of degree 2 are removed as long as the corresponding edges that would be merged are compatible in terms of direction.

Returns: No return value. Called for its side effects.

`metric_graph$set_manual_edge_lengths()`**:** Gets the groups from the data.

Usage:

```
metric_graph$set_manual_edge_lengths(edge_lengths, unit = NULL)
```

Arguments:

edge_lengths edge lengths to be set to the metric graph edges.

unit set or override the edge lengths unit.

Returns: does not return anything. Called for its side effects.

`metric_graph$get_groups()`**:** Gets the groups from the data.

Usage:

```
metric_graph$get_groups(get_cols = FALSE)
```

Arguments:

get_cols Should the names of the columns that created the group variable be returned?

Returns: A vector containing the available groups in the internal data.

`metric_graph$get_PtE()`: Gets PtE from the data.

Usage:

```
metric_graph$get_PtE()
```

Returns: A matrix with two columns, where the first column contains the edge number and the second column contains the distance on edge of the observation locations.

`metric_graph$get_edge_lengths()`: Gets the edge lengths with the corresponding unit.

Usage:

```
metric_graph$get_edge_lengths(unit = NULL)
```

Arguments:

`unit` If non-NULL, changes from `length_unit` from the graph construction to `unit`.

Returns: a vector with the length unit (if the graph was constructed with a length unit).

`metric_graph$get_locations()`: Gets the spatial locations from the data.

Usage:

```
metric_graph$get_locations()
```

Returns: A `data.frame` object with observation locations. If `longlat = TRUE`, the column names are lon and lat, otherwise the column names are x and y.

`metric_graph$observation_to_vertex()`: Adds observation locations as vertices in the graph.

Usage:

```
metric_graph$observation_to_vertex(
  mesh_warning = TRUE,
  verbose = 0,
  tolerance = deprecated()
)
```

Arguments:

`mesh_warning` Display a warning if the graph structure change and the metric graph has a mesh object.

`verbose` Print progress of the steps when adding observations. There are 3 levels of verbose, level 0, 1 and 2. In level 0, no messages are printed. In level 1, only messages regarding important steps are printed. Finally, in level 2, messages detailing all the steps are printed. The default is 1.

`tolerance` **[Deprecated]**. Not used anymore

`share_weights` Should the same weight be shared among the split edges? If `FALSE`, the weights will be removed, and a common weight given by 1 will be given.

Returns: No return value. Called for its side effects.

`metric_graph$edgeweight_to_data()`: Turns edge weights into data on the metric graph

Usage:

```
metric_graph$edgeweight_to_data(
  loc = NULL,
  mesh = FALSE,
  data_loc = FALSE,
  weight_col = NULL,
  add = TRUE,
  data_coords = c("PtE", "spatial"),
  normalized = FALSE,
  tibble = FALSE,
  format = c("tibble", "sf", "sp", "list"),
  verbose = 1,
  suppress_warnings = FALSE,
  return = FALSE
)
```

Arguments:

`loc` A `matrix` or `data.frame` with two columns containing the locations to generate the data from the edge weights. If `data_coords` is 'spatial', the first column must be the x-coordinate of the data, and the second column must be the y-coordinate. If `data_coords` is 'PtE', the first column must be the edge number and the second column must be the distance on edge.

`mesh` Should the data be generated to the mesh locations? In this case, the `loc` argument will be ignored. Observe that the metric graph must have a mesh built for one to use this option. CAUTION: To add edgeweight to data to both the data locations and mesh locations, please, add at the data locations first, then to mesh locations.

`data_loc` Should the data be generated to the data locations? In this case, the `loc` argument will be ignored. Observe that the metric graph must have data for one to use this option. CAUTION: To add edgeweight to data to both the data locations and mesh locations, please, add at the data locations first, then to mesh locations.

`weight_col` Which columns of the edge weights should be turned into data? If `NULL`, all columns will be turned into data.

`add` Should the data generated be added to the metric graph internal data?

`data_coords` To be used only if `mesh` is `FALSE`. It decides which coordinate system to use. If `PtE`, the user must provide `edge_number` and `distance_on_edge`, otherwise if `spatial`, the user must provide `coord_x` and `coord_y`.

`normalized` if TRUE, then the distances in `distance_on_edge` are assumed to be normalized to (0,1). Default FALSE.

`tibble` Should the data be returned in a `tibble` format?

`format` If `return` is `TRUE`, the format of the output: "tibble", "sf", or "sp". Default is "tibble".

`verbose` Print progress of the steps when adding observations. There are 3 levels of verbose, level 0, 1 and 2. In level 0, no messages are printed. In level 1, only messages regarding important steps are printed. Finally, in level 2, messages detailing all the steps are printed. The default is 1.

`suppress_warnings` Suppress warnings related to duplicated observations?

`return` Should the data be returned? If `return_removed` is `TRUE`, only the removed locations will be return (if there is any).

`metric_graph$get_mesh_locations()`**:** Returns a list or a matrix with the mesh locations.

Usage:

```
metric_graph$get_mesh_locations(
  bru = FALSE,
  loc = c(".edge_number", ".distance_on_edge"),
  loc_name = NULL,
  normalized = TRUE
)
```

Arguments:

`bru` Should an 'inlabru'-friendly list be returned?

`loc` If `bru` is set to `TRUE`, the column names of the location variables. The default name is `c('.edge_number', '.distance_on_edge')`.

`loc_name` The name of the location variables. Not needed for `rSPDE` models.

`normalized` If TRUE, then the distances in `distance_on_edge` are assumed to be normalized to (0,1). Default TRUE.

Returns: A list or a matrix containing the mesh locations.

`metric_graph$clear_observations()`**:** Clear all observations from the `metric_graph` object.

Usage:

```
metric_graph$clear_observations()
```

Returns: No return value. Called for its side effects.

`metric_graph$process_data()`**:** Process data to the metric graph data format.

Usage:

```
metric_graph$process_data(
  data = NULL,
  edge_number = "edge_number",
  distance_on_edge = "distance_on_edge",
  coord_x = "coord_x",
  coord_y = "coord_y",
  data_coords = c("PtE", "spatial"),
  group = NULL,
  group_sep = ".",
  normalized = FALSE,
  format = c("tibble", "sf", "sp", "list"),
  duplicated_strategy = "closest",
  include_distance_to_graph = TRUE,
  only_return_removed = FALSE,
  tolerance = max(self$edge_lengths)/2,
  verbose = FALSE,
  suppress_warnings = FALSE,
  Spoints = lifecycle::deprecated(),
  tibble = lifecycle::deprecated()
)
```

Arguments:

`data` A `data.frame` or named list containing the observations. In case of groups, the data.frames for the groups should be stacked vertically, with a column indicating the index of the group. If `data` is not `NULL`, it takes priority over any eventual data in `Spoints`.

`edge_number` Column (or entry on the list) of the `data` that contains the edge numbers. If not supplied, the column with name "edge_number" will be chosen. Will not be used if `Spoints` is not `NULL`.

`distance_on_edge` Column (or entry on the list) of the `data` that contains the edge numbers. If not supplied, the column with name "distance_on_edge" will be chosen. Will not be used if `Spoints` is not `NULL`.

`coord_x` Column (or entry on the list) of the `data` that contains the x coordinate. If not supplied, the column with name "coord_x" will be chosen. Will not be used if `Spoints` is not `NULL` or if `data_coords` is `PtE`.

`coord_y` Column (or entry on the list) of the `data` that contains the y coordinate. If not supplied, the column with name "coord_x" will be chosen. Will not be used if `Spoints` is not `NULL` or if `data_coords` is `PtE`.

`data_coords` It decides which coordinate system to use. If `PtE`, the user must provide `edge_number` and `distance_on_edge`, otherwise if `spatial`, the user must provide `coord_x` and `coord_y`. The option `euclidean` is **[Deprecated]**. Use `spatial` instead.

`group` Vector. If the data is grouped (for example measured at different time points), this argument specifies the columns (or entries on the list) in which the group variables are stored. It will be stored as a single column `.group` with the combined entries.

`group_sep` separator character for creating the new group variable when grouping two or more variables.

`normalized` if TRUE, then the distances in `distance_on_edge` are assumed to be normalized to (0,1). Default FALSE.

`format` Which format should the data be returned? The options are `tibble` for `tidyr::tibble`, `sf` for `POINT`, `sp` for `SpatialPointsDataFrame` and `list` for the internal list format.

`duplicated_strategy` Which strategy to handle observations on the same location on the metric graph (that is, if there are two or more observations projected at the same location). The options are 'closest' and 'jitter'. If 'closest', only the closest observation will be used. If 'jitter', a small perturbation will be performed on the projected observation location. The default is 'closest'.

`include_distance_to_graph` When `data_coord` is 'spatial', should the distance of the observations to the graph be included as a column?

`only_return_removed` Should the removed data (if it exists) when using 'closest' `duplicated_strategy` be returned instead of the processed data?

`tolerance` Parameter to control a warning when adding observations. If the distance of some location and the closest point on the graph is greater than the tolerance, the function will display a warning. This helps detecting mistakes on the input locations when adding new data.

`verbose` If `TRUE`, report steps and times.

`suppress_warnings` Suppress warnings related to duplicated observations?

`Spoints` **[Deprecated]** Use `data` instead.

`tibble` **[Deprecated]** Use `format` instead.

Returns: No return value. Called for its side effects. The observations are stored in the `data` element of the `metric_graph` object.

`metric_graph$add_observations()`: Add observations to the metric graph.

Usage:

```
metric_graph$add_observations(
  data = NULL,
  edge_number = "edge_number",
  distance_on_edge = "distance_on_edge",
  coord_x = "coord_x",
  coord_y = "coord_y",
  data_coords = c("PtE", "spatial"),
  group = NULL,
  group_sep = ".",
  normalized = FALSE,
  clear_obs = FALSE,
  tibble = FALSE,
  tolerance = max(self$edge_lengths)/2,
  duplicated_strategy = "closest",
  include_distance_to_graph = TRUE,
  return_removed = TRUE,
  tolerance_merge = 0,
  merge_strategy = "merge",
  verbose = 1,
  suppress_warnings = FALSE,
  Spoints = lifecycle::deprecated()
)
```

Arguments:

`data` A `data.frame` or named list containing the observations. In case of groups, the data.frames for the groups should be stacked vertically, with a column indicating the index of the group. `data` can also be an `sf` object, a `SpatialPointsDataFrame` object or an `SSN` object. in which case `data_coords` will automatically be spatial, and there is no need to specify the `coord_x` or `coord_y` arguments.

`edge_number` Column (or entry on the list) of the `data` that contains the edge numbers. If not supplied, the column with name "edge_number" will be chosen. Will not be used if `Spoints` is not `NULL`.

`distance_on_edge` Column (or entry on the list) of the `data` that contains the edge numbers. If not supplied, the column with name "distance_on_edge" will be chosen. Will not be used if `Spoints` is not `NULL`.

`coord_x` Column (or entry on the list) of the `data` that contains the x coordinate. If not supplied, the column with name "coord_x" will be chosen. Will not be used if `Spoints` is not `NULL` or if `data_coords` is `PtE`.

`coord_y` Column (or entry on the list) of the `data` that contains the y coordinate. If not supplied, the column with name "coord_x" will be chosen. Will not be used if `Spoints` is not `NULL` or if `data_coords` is `PtE`.

`data_coords` It decides which coordinate system to use. If `PtE`, the user must provide `edge_number` and `distance_on_edge`, otherwise if `spatial`, the user must provide `coord_x` and `coord_y`. The option `euclidean` is **[Deprecated]**. Use `spatial` instead.

`group` Vector. If the data is grouped (for example measured at different time points), this argument specifies the columns (or entries on the list) in which the group variables are stored. It

will be stored as a single column .group with the combined entries.

group_sep separator character for creating the new group variable when grouping two or more variables.

normalized if TRUE, then the distances in distance_on_edge are assumed to be normalized to (0,1). Default FALSE.

clear_obs Should the existing observations be removed before adding the data?

tibble Should the data be returned as a tidyr::tibble?

tolerance Parameter to control a warning when adding observations. If the distance of some location and the closest point on the graph is greater than the tolerance, the function will display a warning. This helps detecting mistakes on the input locations when adding new data.

duplicated_strategy Which strategy to handle observations on the same location on the metric graph (that is, if there are two or more observations projected at the same location). The options are 'closest' and 'jitter'. If 'closest', only the closest observation will be used. If 'jitter', a small perturbation will be performed on the projected observation location. The default is 'closest'.

include_distance_to_graph When data_coord is 'spatial', should the distance of the observations to the graph be included as a column?

return_removed Should the removed data (if it exists) due to being projected to the same place when using 'closest' duplicated_strategy, or due to some merge strategy, be returned?

tolerance_merge tolerance (in edge_length units) for merging points that are very close and are on a common edge. By default, this tolerance is zero, meaning no merges will be performed.

merge_strategy The strategies to handle observations that are within the tolerance. The options are remove, merge, average. The default is merge, in which one of the observations will be chosen, and the remaining will be used to try to fill all columns with non-NA values. The second strategy is remove, meaning that if two observations are within the tolerance one of them will be removed. Finally, average will take the average over the close observations for numerical variables, and will choose one non-NA for non-numerical variables.

verbose Print progress of the steps when adding observations. There are 3 levels of verbose, level 0, 1 and 2. In level 0, no messages are printed. In level 1, only messages regarding important steps are printed. Finally, in level 2, messages detailing all the steps are printed. The default is 1.

suppress_warnings Suppress warnings related to duplicated observations?

Spoints **[Deprecated]** Use data instead.

Returns: No return value. Called for its side effects. The observations are stored in the data element of the metric_graph object.

`metric_graph$mutate_weights()`**:** Use dplyr::mutate function on the internal edge weights object.

Usage:

```
metric_graph$mutate_weights(
  ...,
  .drop_na = FALSE,
  .drop_all_na = TRUE,
  format = "tibble"
)
```

Arguments:

`...` Arguments to be passed to `dplyr::mutate()`.

`.drop_na` Should the rows with at least one NA for one of the columns be removed? DEFAULT is `FALSE`.

`.drop_all_na` Should the rows with all variables being NA be removed? DEFAULT is `TRUE`.

`format` The format of the output: "tibble", "sf", or "sp". Default is "tibble".

Details: A wrapper to use `dplyr::mutate()` on the internal edge weights object and return the result in the requested format.

Returns: A `tidyr::tibble`, `sf` or `sp` object containing the resulting data list after the mutate.

`metric_graph$select_weights()`**:** Use `dplyr::select` function on the internal edge weights object.

Usage:

```
metric_graph$select_weights(
  ...,
  .drop_na = FALSE,
  .drop_all_na = TRUE,
  format = "tibble"
)
```

Arguments:

`...` Arguments to be passed to `dplyr::select()`.

`.drop_na` Should the rows with at least one NA for one of the columns be removed? DEFAULT is `FALSE`.

`.drop_all_na` Should the rows with all variables being NA be removed? DEFAULT is `TRUE`.

`format` The format of the output: "tibble", "sf", or "sp". Default is "tibble".

Details: A wrapper to use `dplyr::select()` on the internal edge weights object and return the result in the requested format.

Returns: A `tidyr::tibble`, `sf` or `sp` object containing the resulting data list after the select.

`metric_graph$filter_weights()`**:** Use `dplyr::filter` function on the internal edge weights object.

Usage:

```
metric_graph$filter_weights(
  ...,
  .drop_na = FALSE,
  .drop_all_na = TRUE,
  format = "tibble"
)
```

Arguments:

`...` Arguments to be passed to `dplyr::filter()`.

`.drop_na` Should the rows with at least one NA for one of the columns be removed? DEFAULT is `FALSE`.

`.drop_all_na` Should the rows with all variables being NA be removed? DEFAULT is `TRUE`.

format The format of the output: "tibble", "sf", or "sp". Default is "tibble".

Details: A wrapper to use dplyr::filter() on the internal edge weights object and return the result in the requested format.

Returns: A tidyr::tibble, sf or sp object containing the resulting data list after the filter.

metric_graph$summarise_weights(): Use dplyr::summarise function on the internal edge weights object grouped by the edge numbers.

Usage:

```
metric_graph$summarise_weights(
  ...,
  .groups = NULL,
  .drop_na = FALSE,
  .drop_all_na = TRUE,
  format = "tibble"
)
```

Arguments:

... Arguments to be passed to dplyr::summarise().

.groups A vector of strings containing the names of the columns to be grouped, when computing the summaries. The default is NULL.

.drop_na Should the rows with at least one NA for one of the columns be removed? DEFAULT is FALSE.

.drop_all_na Should the rows with all variables being NA be removed? DEFAULT is TRUE.

format The format of the output: "tibble", "sf", or "sp". Default is "tibble".

Details: A wrapper to use dplyr::summarise() on the internal edge weights object and return the result in the requested format.

Returns: A tidyr::tibble, sf or sp object containing the resulting data list after the summarise.

metric_graph$drop_na_weights(): Use tidyr::drop_na() function on the internal edge weights object.

Usage:

```
metric_graph$drop_na_weights(..., format = "tibble")
```

Arguments:

... Arguments to be passed to tidyr::drop_na().

format The format of the output: "tibble", "sf", or "sp". Default is "tibble".

Details: A wrapper to use tidyr::drop_na() within the internal edge weights object.

Returns: A tidyr::tibble, sf, or sp object containing the resulting data list after the drop_na.

metric_graph$mutate(): Use dplyr::mutate function on the internal metric graph data object.

Usage:

```
metric_graph$mutate(
  ...,
  .drop_na = FALSE,
  .drop_all_na = TRUE,
  format = "tibble"
)
```

Arguments:

`...` Arguments to be passed to `dplyr::mutate()`.

`.drop_na` Should the rows with at least one NA for one of the columns be removed? DEFAULT is `FALSE`.

`.drop_all_na` Should the rows with all variables being NA be removed? DEFAULT is `TRUE`.

`format` The format of the output: "tibble", "sf", or "sp". Default is "tibble".

Details: A wrapper to use `dplyr::mutate()` within the internal metric graph data object and return the result in the requested format.

Returns: A `tidyr::tibble`, `sf`, or `sp` object containing the resulting data list after the mutate.

`metric_graph$drop_na()`**:** Use `tidyr::drop_na()` function on the internal metric graph data object.

Usage:

```
metric_graph$drop_na(..., format = "tibble")
```

Arguments:

`...` Arguments to be passed to `tidyr::drop_na()`.

`format` The format of the output: "tibble", "sf", or "sp". Default is "tibble".

Details: A wrapper to use `dplyr::drop_na()` within the internal metric graph data object.

Returns: A `tidyr::tibble` object containing the resulting data list after the drop_na.

`metric_graph$select()`**:** Use `dplyr::select` function on the internal metric graph data object.

Usage:

```
metric_graph$select(
  ...,
  .drop_na = FALSE,
  .drop_all_na = TRUE,
  format = "tibble"
)
```

Arguments:

`...` Arguments to be passed to `dplyr::select()`.

`.drop_na` Should the rows with at least one NA for one of the columns be removed? DEFAULT is `FALSE`.

`.drop_all_na` Should the rows with all variables being NA be removed? DEFAULT is `TRUE`.

`format` The format of the output: "tibble", "sf", or "sp". Default is "tibble".

Details: A wrapper to use `dplyr::select()` within the internal metric graph data object. Observe that it is a bit different from directly using `dplyr::select()` since it does not allow to remove the internal positions that are needed for the metric_graph methods to work.

Returns: A `tidyr::tibble` object containing the resulting data list after the selection.

`metric_graph$filter()`**:** Use `dplyr::filter` function on the internal metric graph data object.

Usage:

```
metric_graph$filter(
  ...,
  .drop_na = FALSE,
  .drop_all_na = TRUE,
  format = "tibble"
)
```

Arguments:

`...` Arguments to be passed to `dplyr::filter()`.

`.drop_na` Should the rows with at least one NA for one of the columns be removed? DEFAULT is `FALSE`.

`.drop_all_na` Should the rows with all variables being NA be removed? DEFAULT is `TRUE`.

`format` The format of the output: "tibble", "sf", or "sp". Default is "tibble".

Details: A wrapper to use `dplyr::filter()` within the internal metric graph data object.

Returns: A `tidyr::tibble` object containing the resulting data list after the filter.

`metric_graph$summarise()`**:** Use `dplyr::summarise` function on the internal metric graph data object grouped by the spatial locations and the internal group variable.

Usage:

```
metric_graph$summarise(
  ...,
  .include_graph_groups = FALSE,
  .groups = NULL,
  .drop_na = FALSE,
  .drop_all_na = TRUE,
  format = "tibble"
)
```

Arguments:

`...` Arguments to be passed to `dplyr::summarise()`.

`.include_graph_groups` Should the internal graph groups be included in the grouping variables? The default is `FALSE`. This means that, when summarising, the data will be grouped by the internal group variable together with the spatial locations.

`.groups` A vector of strings containing the names of the columns to be additionally grouped, when computing the summaries. The default is `NULL`.

`.drop_na` Should the rows with at least one NA for one of the columns be removed? DEFAULT is `FALSE`.

`.drop_all_na` Should the rows with all variables being NA be removed? DEFAULT is `TRUE`.

`format` The format of the output: "tibble", "sf", or "sp". Default is "tibble".

Details: A wrapper to use `dplyr::summarise()` within the internal metric graph data object grouped by manually inserted groups (optional), the internal group variable (optional) and the

spatial locations. Observe that if the integral group variable was not used as a grouping variable for the summarise, a new column, called `.group`, will be added, with the same value 1 for all rows.

Returns: A `tidyr::tibble` object containing the resulting data list after the summarise.

`metric_graph$get_data()`**:** Return the internal data with the option to filter by groups.

Usage:

```
metric_graph$get_data(
  group = NULL,
  format = c("tibble", "sf", "sp", "list"),
  drop_na = FALSE,
  drop_all_na = TRUE,
  tibble = deprecated()
)
```

Arguments:

`group` A vector contaning which groups should be returned? The default is `NULL`, which gives the result for the all groups.

`format` Which format should the data be returned? The options are `tibble` for `tidyr::tibble`, `sf` for `POINT`, `sp` for `SpatialPointsDataFrame` and `list` for the internal list format.

`drop_na` Should the rows with at least one NA for one of the columns be removed? DEFAULT is `FALSE`.

`drop_all_na` Should the rows with all variables being NA be removed? DEFAULT is `TRUE`.

`tibble` **[Deprecated]** Use `format` instead.

`metric_graph$setDirectionalWeightFunction()`**:** Define the columns to be used for creating the directional vertex weights. Also possible to supply user defined functions for input and output to create ones own weights.

Usage:

```
metric_graph$setDirectionalWeightFunction(f_in = NULL, f_out = NULL)
```

Arguments:

`f_in` functions for the input vertex (default `w/sum(w)`) uses the columns of name_column

`f_out` functions for the output vertex (deafult `rep(-1,length(w))`) uses the columns of name_column

Details: For more details see paper (that does not exists yet).

Returns: No return value.

`metric_graph$buildDirectionalConstraints()`**:** Build directional ODE constraint matrix from edges.

Usage:

```
metric_graph$buildDirectionalConstraints(alpha = 1)
```

Arguments:

`alpha` how many derivatives the processes has

`weight` weighting for each vertex used in the constraint (E x 2)

Details: Currently not implemented for circles (edges that start and end in the same vertex)

Returns: No return value. Called for its side effects.

`metric_graph$compute_directional_covariance()`: Closed-form covariance of the directional OU model at observation locations or arbitrary points.

Usage:

```
metric_graph$compute_directional_covariance(
  kappa,
  tau,
  PtE = NULL,
  PtE2 = NULL,
  sigma_source = NULL,
  normalized = TRUE,
  cpp = TRUE
)
```

Arguments:

`kappa` Positive OU rate parameter.

`tau` White-noise scale; the stationary variance is `1/(2*kappa*tau^2)`.

`PtE` Evaluation points `(edge_number, distance_on_edge)`.

`PtE2` Optional second point set for cross-covariance.

`sigma_source` Optional source-vertex anchoring variances.

`normalized` Whether point distances are normalized to `[0, 1]`.

`cpp` Use C++ numeric setup and the shared C++ covariance fill when available for oriented in-trees and out-trees.

Returns: A dense covariance matrix.

`metric_graph$buildC()`: Build Kirchoff constraint matrix from edges.

Usage:

```
metric_graph$buildC(alpha = 2, edge_constraint = FALSE)
```

Arguments:

`alpha` the type of constraint (currently only supports 2)

`edge_constraint` if TRUE, add constraints on vertices of degree 1

Details: Circular edges (edges that start and end in the same vertex) are supported: both ends of the edge are constrained at that vertex.

Returns: No return value. Called for its side effects.

`metric_graph$build_mesh()`: Builds mesh object for graph.

Usage:

```
metric_graph$build_mesh(
  h = NULL,
  n = NULL,
  continuous = TRUE,
  continuous.outs = FALSE,
  continuous.deg2 = FALSE
)
```

Arguments:

`h` Maximum distance between mesh nodes (should be provided if n is not provided).

`n` Maximum number of nodes per edge (should be provided if h is not provided).

`continuous` If `TRUE` (default), the mesh contains only one node per vertex. If `FALSE`, each vertex v is split into deg(v) disconnected nodes to allow for the creation of discontinuities at the vertices.

`continuous.outs` If `continuous = FALSE` and `continuous.outs = TRUE`, continuity is assumed for the outgoing edges from each vertex.

`continuous.deg2` If `TRUE`, continuity is assumed at degree 2 vertices.

Details: The mesh is a list with the objects:

- `PtE` The mesh locations excluding the original vertices;
- `V` The verties of the mesh;
- `E` The edges of the mesh;
- `n_e` The number of vertices in the mesh per original edge in the graph;
- `h_e` The mesh width per edge in the graph;
- `ind` The indices of the vertices in the mesh;
- `VtE` All mesh locations including the original vertices.

Returns: No return value. Called for its side effects. The mesh is stored in the `mesh` element of the `metric_graph` object.

`metric_graph$get_version()`**:** Get the version of MetricGraph package used to build the graph

Usage:

```
metric_graph$get_version()
```

Returns: A character string with the version number

`metric_graph$is_disconnected()`**:** Does this graph have more than one connected component? The decomposition is computed and cached at construction time, so this is an O(1) lookup.

Usage:

```
metric_graph$is_disconnected()
```

Returns: `TRUE` if the graph has two or more connected components, `FALSE` otherwise.

`metric_graph$get_components()`**:** Return the connected components of the graph as a list of `metric_graph` objects. For a connected graph this is simply `list(self)`. For a disconnected graph, the components are returned in order of decreasing total edge length and any observations stored on `self` are routed to the appropriate component. The result is cached internally; subsequent calls are O(1) until observations change (which invalidates the cache).

Usage:

```
metric_graph$get_components(verbose = 0)
```

Arguments:

`verbose` Verbosity level passed to the per-component constructors (default `0`).

Returns: A list of `metric_graph` objects.

`metric_graph$get_largest()`: Return only the largest connected component, as a `metric_graph` object. Equivalent to `get_components()[[1]]`, but it builds just that one component instead of all of them, which matters when a graph decomposes into many small pieces and only the main network is of interest. For a connected graph it returns the graph itself. Components are ranked by total edge length, so the result is the same graph `get_components()[[1]]` would return.

Usage:

```
metric_graph$get_largest(verbose = 0)
```

Arguments:

`verbose` Verbosity level passed to the component constructor (default `0`).

Returns: A `metric_graph` object, or `NULL` if the graph has no edges.

`metric_graph$which_component()`: For each spatial point, determine which connected component of the graph it belongs to. The component is the one whose nearest edge is closest in Euclidean distance to the point.

Usage:

```
metric_graph$which_component(XY)
```

Arguments:

`XY` An `n x 2` matrix of spatial coordinates (or a length-2 numeric vector for a single point).

Returns: An integer vector of length `n` with the component index for each point. Indices match those of `get_components()` (i.e. components are sorted by total edge length, descending).

`metric_graph$compute_fem()`: Build mass and stiffness matrices for given mesh object.

Usage:

```
metric_graph$compute_fem(petrov = FALSE)
```

Arguments:

`petrov` Compute Petrov-Galerkin matrices? (default `FALSE`). These are defined as $Cpet_{ij} =< \phi_i, \psi_j >$ and $Gpet_{ij} =< d\phi_i, \psi_j >$, where ψ_i are piecewise constant basis functions on the edges of the mesh.

Details: The function builds: The matrix `C` which is the mass matrix with elements $C_{ij} =< \phi_i, \phi_j >$, the matrix `G` which is the stiffness matrix with elements $G_{ij} =< d\phi_i, d\phi_j >$, the matrix `B` with elements $B_{ij} =< d\phi_i, \phi_j >$, the matrix `D` with elements $D_{ij} = \sum_{v \in V} \phi_i(v)\phi_j(v)$, and the vector with weights $< \phi_i, 1 >$.

Returns: No return value. Called for its side effects. The finite element matrices `C`, `G` and `B` are stored in the `mesh` element in the `metric_graph` object. If `petrov=TRUE`, the corresponding Petrov-Galerkin matrices are stored in `Cpet` and `Gpet`.

`metric_graph$compute_mesh_weights()`: Compute the weights of the mesh nodes.

Usage:

```
metric_graph$compute_mesh_weights()
```

Details: Compute the weights of the mesh nodes.

Returns: No return value. Called for its side effects. The weights are stored in the `mesh` element in the `metric_graph` object.

`metric_graph$mesh_A()`: Deprecated - Computes observation matrix for mesh.
[Deprecated] in favour of `metric_graph$fem_basis()`.

Usage:

```
metric_graph$mesh_A(PtE)
```

Arguments:

`PtE` Locations given as (edge number in graph, normalized location on edge)

Details: For n locations and a mesh with m nodes, `A` is an n x m matrix with elements $A_{ij} = \phi_j(s_i)$.

Returns: The observation matrix.

`metric_graph$fem_basis()`: Computes observation matrix for mesh.

Usage:

```
metric_graph$fem_basis(PtE)
```

Arguments:

`PtE` Locations given as (edge number in graph, normalized location on edge)

Details: For n locations and a mesh with m nodes, `A` is an n x m matrix with elements $A_{ij} = \phi_j(s_i)$.

Returns: The observation matrix.

`metric_graph$VtEfirst()`: Find one edge corresponding to each vertex.

Usage:

```
metric_graph$VtEfirst()
```

Returns: A nV x 2 matrix the first element of the `i`th row is the edge number corresponding to the `i`th vertex and the second value is 0 if the vertex is at the start of the edge and 1 if the vertex is at the end of the edge.

`metric_graph$plot()`: Plots the metric graph.

Usage:

```
metric_graph$plot(
  data = NULL,
  newdata = NULL,
  group = 1,
  type = c("ggplot", "plotly", "mapview"),
  interactive = FALSE,
  vertex_size = 3,
  vertex_color = "black",
  edge_width = 0.3,
  edge_color = "black",
  data_size = 1,
  support_width = 0.5,
  support_color = "gray",
  mesh = FALSE,
  X = NULL,
  X_loc = NULL,
```

```
  p = NULL,
  degree = FALSE,
  direction = FALSE,
  arrow_size = ggplot2::unit(0.25, "inches"),
  edge_weight = NULL,
  edge_width_weight = NULL,
  scale_color_main = ggplot2::scale_color_viridis_c(option = "D"),
  scale_color_weights = ggplot2::scale_color_viridis_c(option = "C"),
  scale_color_degree = ggplot2::scale_color_viridis_d(option = "D"),
 scale_color_weights_discrete = ggplot2::scale_color_viridis_d(option = "C"),
  scale_color_main_discrete = ggplot2::scale_color_viridis_d(option = "C"),
  add_new_scale_weights = TRUE,
  scale_color_mapview = viridis::viridis(100, option = "D"),
  scale_color_weights_mapview = viridis::viridis(100, option = "C"),
  scale_color_weights_discrete_mapview = NULL,
  scale_color_degree_mapview = NULL,
  plotly = deprecated(),
  components = FALSE,
  ...
)
```

Arguments:

`data` Which column of the data to plot? If `NULL`, no data will be plotted.

`newdata` A dataset of class `metric_graph_data`, obtained by any `get_data()`, `mutate()`, `filter()`, `summarise()`, `drop_na()` methods of metric graphs, see the vignette on data manipulation for more details.

`group` If there are groups, which group to plot? If `group` is a number and `newdata` is `NULL`, it will be the index of the group as stored internally and if `newdata` is provided, it will be the index of the group stored in `newdata`. If `group` is a character, then the group will be chosen by its name.

`type` The type of plot to be returned. The options are `ggplot` (the default), that uses `ggplot2`; `plotly` that uses `plot_ly` for 3D plots, which requires the `plotly` package, and `mapview` that uses the `mapview` function, to build interactive plots, which requires the `mapview` package.

`interactive` Only works for 2d plots. If `TRUE`, an interactive plot will be displayed. Unfortunately, `interactive` is not compatible with `edge_weight` if `add_new_scale_weights` is TRUE.

`vertex_size` Size of the vertices.

`vertex_color` Color of vertices.

`edge_width` Line width for edges. If `edge_width_weight` is not `NULL`, this determines the maximum edge width.

`edge_color` Color of edges.

`data_size` Size of markers for data.

`support_width` For 3D plot, width of support lines.

`support_color` For 3D plot, color of support lines.

`mesh` Plot the mesh locations?

`X` Additional values to plot.

X_loc Locations of the additional values in the format (edge, normalized distance on edge).

p Existing objects obtained from 'ggplot2' or 'plotly' to add the graph to

degree Show the degrees of the vertices?

direction Show the direction of the edges? For type == "mapview" the arrows are not shown, only the color of the vertices indicating whether they are problematic or not.

arrow_size The size of the arrows if direction is TRUE.

edge_weight Which column from edge weights to determine the colors of the edges? If NULL edge weights are not plotted. To plot the edge weights when the metric graph edge_weights is a vector instead of a data.frame, simply set to 1. edge_weight is only available for 2d plots. For 3d plots with edge weights, please use the plot_function() method.

edge_width_weight Which column from edge weights to determine the edges widths? If NULL edge width will be determined from edge_width. Currently it is not supported for type = "mapview".

scale_color_main Color scale for the data to be plotted.

scale_color_weights Color scale for the edge weights. Will only be used if add_new_scale_weights is TRUE.

scale_color_degree Color scale for the degrees.

scale_color_weights_discrete Color scale for discrete edge weights. Will only be used if add_new_scale_weights is TRUE.

scale_color_main_discrete Color scale for the data to be plotted, for discrete data.

add_new_scale_weights Should a new color scale for the edge weights be created?

scale_color_mapview Color scale to be applied for data when type = "mapview".

scale_color_weights_mapview Color scale to be applied for edge weights when type = "mapview".

scale_color_weights_discrete_mapview Color scale to be applied for degrees when type = "mapview". If NULL RColorBrewer::brewer.pal(n = n_weights, "Set1") will be used where n_weights is the number of different degrees.

scale_color_degree_mapview Color scale to be applied for degrees when type = "mapview". If NULL RColorBrewer::brewer.pal(n = n_degrees, "Set1") will be used where n_degrees is the number of different degrees.

plotly **[Deprecated]** Use type instead.

components Color the connected components separately. Either FALSE (the default; the graph is plotted as-is), TRUE (each component is drawn in a randomly-chosen color), or an n x 3 numeric matrix of RGB values in [0, 1] (one row per component, indices matching get_components()).

... Additional arguments to pass to ggplot() or plot_ly()

Returns: A plot_ly (if type = "plotly") or ggplot object.

`metric_graph$plot_connections()`**:** Plots the connections in the graph

Usage:

```
metric_graph$plot_connections()
```

Returns: No return value. Called for its side effects.

`metric_graph$is_tree()`**:** Checks if the graph is a tree (without considering directions)

Usage:

```
metric_graph$is_tree()
```

Returns: TRUE if the graph is a tree and FALSE otherwise.

`metric_graph$plot_function()`: Plots continuous function on the graph.

Usage:

```
metric_graph$plot_function(
  data = NULL,
  newdata = NULL,
  group = 1,
  type = c("ggplot", "plotly", "mapview"),
  continuous = TRUE,
  interpolate_plot = TRUE,
  edge_weight = NULL,
  vertex_size = 5,
  vertex_color = "black",
  edge_width = 1,
  edge_color = "black",
  line_width = NULL,
  line_color = "rgb(0,0,200)",
  scale_color = ggplot2::scale_color_viridis_c(option = "D"),
  scale_color_mapview = viridis::viridis(100, option = "D"),
  support_width = 0.5,
  support_color = "gray",
  mapview_caption = "Function",
  p = NULL,
  plotly = deprecated(),
  improve_plot = deprecated(),
  X = deprecated(),
  ...
)
```

Arguments:

`data` Which column of the data to plot? If `NULL`, no data will be plotted.

`newdata` A dataset of class `metric_graph_data`, obtained by any `get_data()`, `mutate()`, `filter()`, `summarise()`, `drop_na()` methods of metric graphs, see the vignette on data manipulation for more details.

`group` If there are groups, which group to plot? If `group` is a number, it will be the index of the group as stored internally. If `group` is a character, then the group will be chosen by its name.

`type` The type of plot to be returned. The options are `ggplot` (the default), that uses `ggplot2`; `plotly` that uses `plot_ly` for 3D plots, which requires the `plotly` package, and `mapview` that uses the `mapview` function, to build interactive plots, which requires the `mapview` package.

`continuous` Should continuity be assumed when the plot uses `newdata`?

`interpolate_plot` Should the values to be plotted be interpolated?

`edge_weight` Which column from edge weights to plot? If `NULL` edge weights are not plotted. To plot the edge weights when the metric graph `edge_weights` is a vector instead of a `data.frame`, simply set to 1.

vertex_size Size of the vertices.

vertex_color Color of vertices.

edge_width Width for edges.

edge_color For 3D plot, color of edges.

line_width For 3D plot, line width of the function curve.

line_color Color of the function curve.

scale_color Color scale to be used for data and weights.

scale_color_mapview Color scale to be applied for data when type = "mapview".

support_width For 3D plot, width of support lines.

support_color For 3D plot, color of support lines.

mapview_caption Caption for the function if type = "mapview".

p Previous plot to which the new plot should be added.

plotly **[Deprecated]** Use type instead.

improve_plot **[Deprecated]** Use interpolate instead. There is no need to use it to improve the edges.

X **[Deprecated]** Use newdata instead.

... Additional arguments for ggplot() or plot_ly()

Returns: Either a ggplot (if plotly = FALSE) or a plot_ly object.

`metric_graph$plot_movie()`**:** Plots a movie of a continuous function evolving on the graph.

Usage:

```
metric_graph$plot_movie(
  X,
  type = "plotly",
  vertex_size = 5,
  vertex_color = "black",
  edge_width = 1,
  edge_color = "black",
  line_width = NULL,
  line_color = "rgb(0,0,200)",
  ...
)
```

Arguments:

X A m x T matrix where the ith column represents the function at the ith time, evaluated at the mesh locations.

type Type of plot. Either "plotly" or "ggplot".

vertex_size Size of the vertices.

vertex_color Color of vertices.

edge_width Width for edges.

edge_color For 3D plot, color of edges.

line_width For 3D plot, line width of the function curve.

line_color Color of the function curve.

... Additional arguments for ggplot or plot_ly.

Returns: Either a ggplot (if plotly=FALSE) or a plot_ly object.

`metric_graph$add_mesh_observations()`: Add observations on mesh to the object.

Usage:

```
metric_graph$add_mesh_observations(data = NULL, group = NULL)
```

Arguments:

`data` A data.frame or named list containing the observations. In case of groups, the data.frames for the groups should be stacked vertically, with a column indicating the index of the group. If data_frame is not NULL, it takes priority over any eventual data in Spoints.

`group` If the data_frame contains groups, one must provide the column in which the group indices are stored.

Returns: No return value. Called for its side effects. The observations are stored in the data element in the metric_graph object.

`metric_graph$get_initial_graph()`: Returns a copy of the initial metric graph.

Usage:

```
metric_graph$get_initial_graph()
```

Returns: A metric_graph object.

`metric_graph$update_graph()`: Update an older version metric graph to the current package version.

Usage:

```
metric_graph$update_graph(verbose = TRUE)
```

Arguments:

`verbose` Print progress messages. Default is TRUE.

Details: This function creates a new metric_graph object from an existing one, preserving all data, edge weights, mesh, distances, and other components while ensuring compatibility with the current package version. The new graph is created with minimal changes by disabling merges and other transformations.

Returns: A new metric_graph object compatible with the current package version.

`metric_graph$coordinates()`: Convert between locations on the graph and Euclidean coordinates.

Usage:

```
metric_graph$coordinates(PtE = NULL, XY = NULL, normalized = TRUE)
```

Arguments:

`PtE` Matrix with locations on the graph (edge number and normalized position on the edge).

`XY` Matrix with locations in Euclidean space

`normalized` If TRUE, it is assumed that the positions in PtE are normalized to (0,1), and the object returned if XY is specified contains normalized locations.

Returns: If PtE is specified, then a matrix with Euclidean coordinates of the locations is returned. If XY is provided, then a matrix with the closest locations on the graph is returned. Gets the edge weights data.frame If the edge weights are given as vectors, should the result be returned as a data.frame? A vector or data.frame containing the edge weights. data List containing data on the metric graph.

`metric_graph$clone()`: The objects of this class are cloneable with this method.

Usage:

```
metric_graph$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
edge1 <- rbind(c(0, 0), c(2, 0))
edge2 <- rbind(c(2, 0), c(1, 1))
edge3 <- rbind(c(1, 1), c(0, 0))
edges <- list(edge1, edge2, edge3)
graph <- metric_graph$new(edges)
graph$plot()
```

---

metric_graph_from_osm *Build a metric_graph directly from an OpenStreetMap query*

Description

Convenience wrapper that fetches OSM data with `fetch_osm()` and builds a metric_graph from the returned linestrings in one call. `longlat = TRUE` is set automatically because OSM coordinates are always in WGS84 latitude / longitude. Other graph-construction arguments (`tolerance`, `perform_merges`, ...) are forwarded to metric_graph's `$new()` method without overriding its defaults.

Usage

```
metric_graph_from_osm(
  bbox,
  key = "highway",
  value = NULL,
  endpoint = "https://overpass-api.de/api/interpreter",
  timeout = 180,
  user_agent = "MetricGraph-R-package",
  cache_path = NULL,
  retries = 2,
  ...
)
```

Arguments

bbox	A bounding box, passed to `osmdata::opq()`. May be a length-4 numeric vector `c(xmin, ymin, xmax, ymax)`, an `sf::bbox` object, or any other form accepted by `osmdata::opq()`.

key	OSM key to filter on, e.g. "highway". Passed to osmdata::add_osm_feature(). Use NULL to skip the feature filter and fetch everything in the bbox.
value	Optional character vector of values for key, e.g. c("motorway", "motorway_link").
endpoint	Overpass endpoint URL, or a character vector of URLs that are tried in order. Defaults to the main overpass-api.de endpoint. Mirror URLs (e.g. https://overpass.kumi.systems work too.
timeout	Server-side query timeout in seconds. Forwarded to both osmdata::opq() and the HTTP request.
user_agent	String sent as the User-Agent HTTP header.
cache_path	Optional path. If provided and the file already exists, the download is skipped and the cached response is parsed instead. If provided and the file does not exist, the response is saved there for re-use.
retries	Number of times a request to each endpoint is retried after a transient failure (HTTP 429 or 5xx, or a connection error), with exponential backoff between attempts.
...	Forwarded to metric_graph's $new(). Use this to pass tolerance, perform_merges, check_connected, which_longlat, etc.

Value

A metric_graph object built from the OSM ways.

See Also

fetch_osm() for the lower-level fetch + parse step.

Examples

```
## Not run:
## Build a freeway graph for a small bbox around San Jose, splitting
## at intersections via the edge_edge tolerance.
g <- metric_graph_from_osm(
  bbox           = c(-122.10, 37.22, -121.80, 37.45),
  value          = c("motorway", "motorway_link"),
  perform_merges = TRUE,
  tolerance      = list(edge_edge = 1e-5)
)
g$plot()

## End(Not run)
```

mutate.metric_graph_data *A version of* dplyr::mutate() *function for datasets on metric graphs*

Description

Applies dplyr::mutate() function for datasets obtained from a metric graph object.

Usage

```
## S3 method for class 'metric_graph_data'
mutate(.data, ...)
```

Arguments

.data	The data list or tidyr::tibble obtained from a metric graph object.
...	Additional parameters to be passed to dplyr::mutate().

Value

A tidyr::tibble with the resulting selected columns.

pems *Traffic speed data from San Jose, California*

Description

Data set of traffic speed observations on highways in the city of San Jose, California.

Usage

```
pems
```

Format

pems**:**

A list with two elements:

edges A list object containing the coordinates of the road segments.

data Locations of the observations on the road segments as a data.frame with 325 rows and 3 columns. The first column indicates the edge number, the second column indicates the distance on edge of the position, and the third column indicates the average speed observed.

Source

https://www.openstreetmap.org

https://github.com/spbu-math-cs/Graph-Gaussian-Processes/blob/main/examples/data/PEMS.zip

References

Chen, C., K. Petty, A. Skabardonis, P. Varaiya, and Z. Jia (2001). Freeway performance measurement system: mining loop detector data. Transportation Research Record 1748(1), 96-102.

OpenStreetMap contributors (2017). Planet dump retrieved from https://planet.osm.org. https://www.openstreetmap.org.

`pems_repl` *Traffic speed data with replicates from San Jose, California*

Description

Data set of traffic speed observations on highways in the city of San Jose, California.

Usage

```
pems_repl
```

Format

`pems_repl`**:**

A list with two elements:

edges A `list` object containing the coordinates of the road segments.

data Locations of the observations on the road segments as a `data.frame` with 325 rows and 4 columns. The first column indicates the observed speed, the second column indicates the edge number, the third column indicates the distance on edge of the position, and the fourth column indicates the replicate number.

Source

https://www.openstreetmap.org

https://github.com/spbu-math-cs/Graph-Gaussian-Processes/blob/main/examples/data/PEMS.zip

References

Chen, C., K. Petty, A. Skabardonis, P. Varaiya, and Z. Jia (2001). Freeway performance measurement system: mining loop detector data. Transportation Research Record 1748(1), 96-102.

OpenStreetMap contributors (2017). Planet dump retrieved from https://planet.osm.org. https://www.openstreetmap.org.

`plot.graph_bru_pred` *Plot of predicted values with 'inlabru'*

Description

Auxiliary function to obtain plots of the predictions of the field using 'inlabru'.

Usage

```
## S3 method for class 'graph_bru_pred'
plot(x, y = NULL, vertex_size = 0, ...)
```

Arguments

`x`	A predicted object obtained with the `predict` method.
`y`	Not used.
`vertex_size`	Size of the vertices.
`...`	Additional parameters to be passed to plot_function.

Value

A 'ggplot2' object.

`plot.graph_bru_proc_pred`
Plot of processed predicted values with 'inlabru'

Description

Auxiliary function to obtain plots of the processed predictions of the field using 'inlabru'.

Usage

```
## S3 method for class 'graph_bru_proc_pred'
plot(x, y = NULL, vertex_size = 0, ...)
```

Arguments

`x`	A processed predicted object obtained with the `process_rspde_predictions` function.
`y`	Not used.
`vertex_size`	Size of the vertices.
`...`	Additional parameters to be passed to plot_function.

Value

A 'ggplot2' object.

posterior_crossvalidation.graph_lme *Cross-validation for* graph_lme *models assuming observations at the vertices of metric graphs*

Description

This function performs cross-validation by computing predictions for test data using either the posterior distribution from a fitted model (pseudo-CV) or by refitting the model for each fold (true CV).

Usage

```
## S3 method for class 'graph_lme'
posterior_crossvalidation(
  object,
  scores = c("logscore", "crps", "scrps", "mae", "rmse"),
  mode = "k-fold",
  k = 10,
  percentage = 20,
  number_folds = 10,
  train_test_indices = NULL,
  true_CV = FALSE,
  factor = 1,
  tibble = TRUE,
  parallel_folds = FALSE,
  parallel_fitting = FALSE,
  n_cores = parallel::detectCores() - 1,
  print = FALSE,
  seed = NULL,
  return_indices = FALSE,
  use_precomputed = TRUE,
  ...
)
```

Arguments

object	A fitted model using the graph_lme() function. Lists of fitted models are handled by the list method of the generic rSPDE::posterior_crossvalidation(), which can also mix graph_lme and rspde_lme fits.
scores	A vector of scores to compute. The options are "logscore", "crps", "scrps", "mae", and "rmse". By default, all scores are computed.
mode	Cross-validation mode. Options are "k-fold", "loo" (leave-one-out), or "lpo" (leave-percentage-out). Default is "k-fold".
k	Number of folds for k-fold cross-validation. Default is 10.

percentage	The percentage (from 1 to 99) of the data to be used to train the model. Will only be used if mode is "lpo". Default is 80.
number_folds	Number of folds to be done if mode is "lpo". Default is 10.
train_test_indices	Optional list containing train and test indices for each fold. If provided, k, mode, and percentage are ignored.
true_CV	Logical indicating whether to refit the model for each fold (TRUE) or use the posterior distribution from the fitted model (FALSE). Default is FALSE.
factor	Which factor to multiply the scores. The default is 1.
tibble	Return the scores as a tidyr::tibble()
parallel_folds	Logical indicating whether to run computations in parallel across folds. Default is FALSE.
parallel_fitting	Logical indicating whether to run model fitting in parallel. Default is FALSE.
n_cores	Number of cores to use for parallel computation. Default is parallel::detectCores() - 1.
print	Logical indicating whether to print progress of which fold is being processed. Default is FALSE.
seed	Random seed for reproducibility in fold creation. Default is NULL.
return_indices	Logical indicating whether to return the train/test indices used. Default is FALSE.
use_precomputed	Logical indicating whether to use precomputation for faster CV. Default is TRUE.
...	Currently not used.

Value

Vector with the posterior expectations and variances as well as mean absolute error (MAE), root mean squared errors (RMSE), and three negatively oriented proper scoring rules: log-score, CRPS, and scaled CRPS.

---

posterior_crossvalidation_loo

Leave-one-out pseudo-crossvalidation for graph_lme *models assuming observations at the vertices of metric graphs*

Description

This function performs pseudo-crossvalidation by computing leave-one-out predictions using the posterior distribution from a fitted model. In pseudo-crossvalidation, the model parameters are kept fixed at the values estimated from the full dataset (those provided in the object), rather than re-estimating them for each fold.

Usage

```
posterior_crossvalidation_loo(
  object,
  factor = 1,
  tibble = TRUE,
  which_repl = NULL
)
```

Arguments

object	A fitted model using the graph_lme() function or a named list of fitted objects using the graph_lme() function.
factor	Which factor to multiply the scores. The default is 1.
tibble	Return the scores as a tidyr::tibble()
which_repl	Which replicates to consider?

Value

Vector with the posterior expectations and variances as well as mean absolute error (MAE), root mean squared errors (RMSE), and three negatively oriented proper scoring rules: log-score, CRPS, and scaled CRPS.

---

precompute_lgcp_graph *Precompute expensive quantities for efficient LGCP model fitting*

Description

This function precomputes the computationally expensive quantities needed for fitting log-Gaussian Cox process (LGCP) models on metric graphs. It enables efficient refitting of multiple models with different formulas while reusing the same spatial structure, integration points, and covariates. This is particularly beneficial for model selection, cross-validation, or sensitivity analysis scenarios.

Usage

```
precompute_lgcp_graph(
  graph,
  resp_variable_name,
  model_name,
  spde_model,
  covariates = NULL,
  interpolate = TRUE,
  manual_integration_points = NULL,
  manual_covariates = NULL,
  use_current_mesh = TRUE,
  new_h = NULL,
  new_n = NULL,
```

```
  repl = ".all",
  repl_col = ".group",
  clone_graph = TRUE
)
```

Arguments

graph	A metric_graph object containing the network structure and point pattern data. Must have observations added via add_observations().
resp_variable_name	Character. Name of the response variable (typically binary: 0 or 1) in the graph data that represents point occurrences.
model_name	Character. Name to be used for the spatial field in INLA formulas (e.g., "field" for f(field, model = spde_model)).
spde_model	An SPDE model object of class inla_metric_graph_spde (from graph_spde()) or rspde_metric_graph (from rspde.metric_graph()).
covariates	Character vector. Names of covariates to include in the precomputation. Only covariates listed here can be used in subsequent model fits with the precomputed data.
interpolate	Logical. If TRUE (default), interpolate covariate values from graph data to integration points. If FALSE, use manual_covariates.
manual_integration_points	Data frame with columns edge_number, distance_on_edge, and E (integration weights). If NULL, automatic integration points are created from the mesh or specified parameters.
manual_covariates	Data frame or named list containing covariate values at integration points. Required when interpolate = FALSE. Must include a .group column for replicates if using replicated data.
use_current_mesh	Logical. If TRUE (default), use the existing mesh in graph$mesh$VtE as integration points. If FALSE or no mesh exists, create a new mesh using new_h or new_n.
new_h	Numeric. Mesh resolution for creating a new mesh when use_current_mesh = FALSE. Smaller values create finer meshes.
new_n	Integer. Alternative to new_h, specifies the approximate number of mesh nodes for the new mesh.
repl	Character vector or ".all". Specifies which replicates to include in the model. Use ".all" to include all available replicates.
repl_col	Character. Name of the column in the graph data that contains replicate identifiers. Default is ".group".
clone_graph	Logical. If TRUE (default), clone the graph to avoid modifying the original object. If FALSE, work directly on the original graph (faster but modifies the input).

Details

This function is most beneficial when:

- Fitting multiple models with different covariate combinations
- Performing model selection or cross-validation
- Using exact SPDE models (which have higher setup costs)
- Working with large graphs or complex spatial structures

The speedup typically becomes apparent when fitting 3 or more models, with greater benefits for more complex spatial structures and exact SPDE models.

Value

A list of class `"precomputed_lgcp"` containing:

graph The modified metric_graph object with integration points

covariates Vector of available covariate names

stk Precomputed INLA stack object

resp_variable_name Name of the response variable

aux_spde_model The SPDE model object prepared for the graph

type_model Type of SPDE model ("exact" or "rational")

model_name Name of the spatial field for formulas

Note

- All covariates you plan to use must be specified in the initial precomputation
- The spatial structure (mesh, integration points) is fixed during precomputation
- Manual covariate values should correspond to mesh nodes in `graph$mesh$VtE` when `interpolate = FALSE`

See Also

`lgcp_graph` for fitting LGCP models with precomputed data, `graph_lgcp_sim` for simulating LGCP data, `graph_spde` for exact SPDE models, `spde_metric_graph_result` for extracting spatial parameter estimates

Examples

```
## Not run:
# Setup: Create graph with mesh and add point pattern data
graph <- metric_graph$new()
graph$build_mesh(h = 0.1)

# Add point pattern data with covariates
point_data <- data.frame(
  y = 1,  # All observed locations have y = 1
  edge_number = c(1, 2, 3, 1, 2),
```

```
  distance_on_edge = c(0.1, 0.3, 0.8, 0.9, 0.2),
  elevation = c(100, 150, 200, 120, 180),
  temperature = c(15, 12, 8, 14, 10)
)
graph$add_observations(point_data, normalized = TRUE)

# Create SPDE model
spde_model <- graph_spde(graph, alpha = 1)

# Precompute for multiple model fitting scenarios
precomputed_data <- precompute_lgcp_graph(
  graph = graph,
  resp_variable_name = "y",
  model_name = "field",
  spde_model = spde_model,
  covariates = c("elevation", "temperature"),
  use_current_mesh = TRUE
)

# Now fit multiple models efficiently
# Model selection: compare different covariate combinations
fit1 <- lgcp_graph(y ~ elevation + f(field, model = spde_model),
                   graph = graph, precomputed_data = precomputed_data)

fit2 <- lgcp_graph(y ~ temperature + f(field, model = spde_model),
                   graph = graph, precomputed_data = precomputed_data)

fit3 <- lgcp_graph(y ~ elevation + temperature + f(field, model = spde_model),
                   graph = graph, precomputed_data = precomputed_data)

# Compare models using log marginal likelihood
c(fit1$mlik[1], fit2$mlik[1], fit3$mlik[1])

# Using manual covariates (exact values at mesh nodes)
manual_covs <- data.frame(
  elevation = graph$mesh$VtE[,1] * 50 + 100,  # Synthetic elevation
  temperature = 20 - graph$mesh$VtE[,1] * 10, # Synthetic temperature
  .group = 1
)

precomputed_manual <- precompute_lgcp_graph(
  graph = graph,
  resp_variable_name = "y",
  model_name = "field",
  spde_model = spde_model,
  covariates = c("elevation", "temperature"),
  manual_covariates = manual_covs,
  interpolate = FALSE
)

# For maximum performance (modifies original graph)
precomputed_fast <- precompute_lgcp_graph(
  graph = graph,
```

```
  resp_variable_name = "y",
  model_name = "field",
  spde_model = spde_model,
  covariates = c("elevation", "temperature"),
  clone_graph = FALSE
)

# Example with replicates
replicate_data <- data.frame(
  y = 1,
  edge_number = c(1, 2, 1, 3, 2, 3),
  distance_on_edge = c(0.2, 0.4, 0.7, 0.1, 0.8, 0.6),
  elevation = c(110, 160, 130, 190, 170, 210),
  replicate_id = c(1, 1, 1, 2, 2, 2)
)

graph$clear_observations()
graph$add_observations(replicate_data, normalized = TRUE, group = "replicate_id")

precomputed_reps <- precompute_lgcp_graph(
  graph = graph,
  resp_variable_name = "y",
  model_name = "field",
  spde_model = spde_model,
  covariates = "elevation",
  repl = ".all",
  repl_col = "replicate_id"
)

fit_reps <- lgcp_graph(y ~ elevation + f(field, model = spde_model,
                                         replicate = field.repl),
                       graph = graph, precomputed_data = precomputed_reps)

## End(Not run)
```

`predict.graph_lme` *Prediction for a mixed effects regression model on a metric graph*

Description

Prediction for a mixed effects regression model on a metric graph

Usage

```
## S3 method for class 'graph_lme'
predict(
  object,
  newdata = NULL,
  mesh = FALSE,
```

```
  mesh_h = 0.01,
  which_repl = NULL,
  compute_variances = FALSE,
  compute_pred_variances = FALSE,
  posterior_samples = FALSE,
  pred_samples = FALSE,
  n_samples = 100,
  edge_number = "edge_number",
  distance_on_edge = "distance_on_edge",
  normalized = FALSE,
  no_nugget = FALSE,
  return_as_list = FALSE,
  return_original_order = TRUE,
  check_euclidean = TRUE,
  advanced_options = list(),
  ...,
  data = deprecated()
)
```

Arguments

object	The fitted object with the graph_lme() function.
newdata	A data.frame or a list containing the covariates, the edge number and the distance on edge for the locations to obtain the prediction. Observe that you should not provide the locations for each replicate. Only a single set of locations and covariates, and the predictions for the different replicates will be obtained for this same set of locations.
mesh	Obtain predictions for mesh nodes? The graph must have a mesh and should not have covariates.
mesh_h	If the graph does not have a mesh, one will be created with this value of 'h'.
which_repl	Which replicates to obtain the prediction. If NULL predictions will be obtained for all replicates. Default is NULL.
compute_variances	Set to TRUE to compute the kriging variances.
compute_pred_variances	Set to TRUE to compute the prediction variances. Will only be computed if newdata is NULL.
posterior_samples	If TRUE, posterior samples for the random effect will be returned.
pred_samples	If TRUE, prediction samples for the response variable will be returned. Will only be computed if newdata is NULL.
n_samples	Number of samples to be returned. Will only be used if sampling is TRUE.
edge_number	Name of the variable that contains the edge number, the default is edge_number.
distance_on_edge	Name of the variable that contains the distance on edge, the default is distance_on_edge.
normalized	Are the distances on edges normalized?

no_nugget	Should the prediction be carried out without the nugget?
return_as_list	Should the means of the predictions and the posterior samples be returned as a list, with each replicate being an element?
return_original_order	Should the results be return in the original (input) order or in the order inside the graph?
check_euclidean	Check if the graph used to compute the resistance distance has Euclidean edges? The graph used to compute the resistance distance has the observation locations as vertices.
advanced_options	Advanced options for internal use only. This parameter is intended to be used by the cross-validation function and should not be used otherwise.
...	Not used.
data	**[Deprecated]** Use newdata instead.

Value

A list with elements mean, which contains the means of the predictions, fe_mean, which is the prediction for the fixed effects, re_mean, which is the prediction for the random effects, variance (if compute_variance is TRUE), which contains the posterior variances of the random effects, samples (if posterior_samples is TRUE), which contains the posterior samples.

---

predict.inla_metric_graph_spde

Predict method for 'inlabru' fits on Metric Graphs

Description

Auxiliar function to obtain predictions of the field using 'inlabru'.

Usage

```
## S3 method for class 'inla_metric_graph_spde'
predict(
  object,
  cmp,
  bru_fit,
  newdata = NULL,
  formula = NULL,
  data_coords = c("PtE", "euclidean"),
  normalized = TRUE,
  repl = NULL,
  repl_col = NULL,
  group = NULL,
  group_col = NULL,
```

```
  n.samples = 100,
  seed = 0L,
  probs = c(0.025, 0.5, 0.975),
  return_original_order = TRUE,
  num.threads = NULL,
  include = NULL,
  exclude = NULL,
  drop = FALSE,
  tolerance_merge = 1e-05,
  ...,
  data = deprecated()
)
```

Arguments

object	An inla_metric_graph_spde object built with the graph_spde() function.
cmp	The 'inlabru' component used to fit the model.
bru_fit	A fitted model using 'inlabru' or 'INLA'.
newdata	A data.frame of covariates needed for the prediction. The locations must be normalized PtE.
formula	A formula where the right hand side defines an R expression to evaluate for each generated sample. If NULL, the latent and hyperparameter states are returned as named list elements. See Details for more information.
data_coords	It decides which coordinate system to use. If PtE, the user must provide the locations as a data frame with the first column being the edge number and the second column as the distance on edge, otherwise if euclidean, the user must provide a data frame with the first column being the x Euclidean coordinates and the second column being the y Euclidean coordinates.
normalized	if TRUE, then the distances in distance on edge are assumed to be normalized to (0,1). Default TRUE. Will not be used if data_coords is euclidean.
repl	Which replicates? If there is no replicates, one can set repl to NULL. If one wants all replicates, then one sets to repl to .all.
repl_col	Column containing the replicates. If the replicate is the internal group variable, set the replicates to ".group". If not replicates, set to NULL.
group	Which groups? If there is no groups, one can set group to NULL. If one wants all groups, then one sets to group to .all.
group_col	Which "column" of the data contains the group variable?
n.samples	Integer setting the number of samples to draw in order to calculate the posterior statistics. The default is rather low but provides a quick approximate result.
seed	Random number generator seed passed on to inla.posterior.sample()
probs	A numeric vector of probabilities with values in the standard unit interval to be passed to stats::quantile
return_original_order	Should the predictions be returned in the original order?

num.threads	Specification of desired number of threads for parallel computations. Default NULL, leaves it up to 'INLA'. When seed != 0, overridden to "16:1"
include	Character vector of component labels that are needed by the predictor expression; Default: NULL (include all components that are not explicitly excluded)
exclude	Character vector of component labels that are not used by the predictor expression. The exclusion list is applied to the list as determined by the include parameter; Default: NULL (do not remove any components from the inclusion list)
drop	logical; If keep=FALSE, data is a SpatialDataFrame, and the prediciton summary has the same number of rows as data, then the output is a SpatialDataFrame object. Default FALSE.
tolerance_merge	Tolerance for merging prediction points into original points to increase stability.
...	Additional arguments passed on to `inla.posterior.sample()`.
data	**[Deprecated]** Use `newdata` instead.

Value

A list with predictions.

`predict.rspde_metric_graph` *Predict method for 'inlabru' fits on Metric Graphs for 'rSPDE' models*

Description

Auxiliar function to obtain predictions of the field using 'inlabru' and 'rSPDE'.

Usage

```
## S3 method for class 'rspde_metric_graph'
predict(
  object,
  cmp,
  bru_fit,
  newdata = NULL,
  formula = NULL,
  data_coords = c("PtE", "euclidean"),
  normalized = TRUE,
  n.samples = 100,
  seed = 0L,
  probs = c(0.025, 0.5, 0.975),
  num.threads = NULL,
  include = NULL,
  exclude = NULL,
```

```
  drop = FALSE,
  ...,
  data = deprecated()
)
```

Arguments

object	An rspde_metric_graph object built with the rspde.metric_graph() function.
cmp	The 'inlabru' component used to fit the model.
bru_fit	A fitted model using 'inlabru' or 'INLA'.
newdata	A data.frame of covariates needed for the prediction. The locations must be normalized PtE.
formula	A formula where the right hand side defines an R expression to evaluate for each generated sample. If NULL, the latent and hyperparameter states are returned as named list elements. See Details for more information.
data_coords	It decides which coordinate system to use. If PtE, the user must provide the locations as a data frame with the first column being the edge number and the second column as the distance on edge, otherwise if euclidean, the user must provide a data frame with the first column being the x Euclidean coordinates and the second column being the y Euclidean coordinates.
normalized	if TRUE, then the distances in distance on edge are assumed to be normalized to (0,1). Default TRUE. Will not be used if data_coords is euclidean.
n.samples	Integer setting the number of samples to draw in order to calculate the posterior statistics. The default is rather low but provides a quick approximate result.
seed	Random number generator seed passed on to inla.posterior.sample
probs	A numeric vector of probabilities with values in the standard unit interval to be passed to stats::quantile.
num.threads	Specification of desired number of threads for parallel computations. Default NULL, leaves it up to 'INLA'. When seed != 0, overridden to "16:1"
include	Character vector of component labels that are needed by the predictor expression; Default: NULL (include all components that are not explicitly excluded)
exclude	Character vector of component labels that are not used by the predictor expression. The exclusion list is applied to the list as determined by the include parameter; Default: NULL (do not remove any components from the inclusion list)
drop	logical; If keep=FALSE, data is a SpatialDataFrame, and the prediciton summary has the same number of rows as data, then the output is a SpatialDataFrame object. Default FALSE.
...	Additional arguments passed on to inla.posterior.sample.
data	**[Deprecated]** Use newdata instead.

Value

A list with predictions.

process_rspde_predictions *Process predictions of* rspde_metric_graph *objects obtained by using* inlabru

Description

Auxiliar function to transform the predictions of the field into a plot friendly object.

Usage

```
process_rspde_predictions(pred, graph, PtE = NULL)
```

Arguments

pred	The predictions of the field obtained by using inlabru
graph	The original metric_graph object in which the predictions were obtained.
PtE	Normalized locations of the points on the edge.

Value

A list with predictions.

psp.to.graph *Convert a* psp *object to a metric graph object*

Description

This function converts a psp object (from the spatstat package) into a metric graph object.

Usage

```
psp.to.graph(psp.object)
```

Arguments

psp.object	A psp object to be converted.

Value

A metric graph object with edges defined by the segments in the psp object.

`sample_spde` *Samples a Whittle-Matérn field on a metric graph*

Description

Obtains samples of a Whittle-Matérn field on a metric graph.

Usage

```
sample_spde(
  kappa,
  tau,
  range,
  sigma,
  sigma_e = 0,
  alpha = 1,
  directional = FALSE,
  graph,
  PtE = NULL,
  type = "manual",
  posterior = FALSE,
  nsim = 1,
  method = c("conditional", "Q"),
  BC = 1
)
```

Arguments

`kappa`	Range parameter.
`tau`	Precision parameter.
`range`	Practical correlation range parameter.
`sigma`	Marginal standard deviation parameter.
`sigma_e`	Standard deviation of the measurement noise.
`alpha`	Smoothness parameter.
`directional`	should we use directional model currently only for alpha=1
`graph`	A `metric_graph` object.
`PtE`	Matrix with locations (edge, normalized distance on edge) where the samples should be generated.
`type`	If "manual" is set, then sampling is done at the locations specified in `PtE`. Set to "mesh" for simulation at mesh nodes, and to "obs" for simulation at observation locations.
`posterior`	Sample conditionally on the observations?
`nsim`	Number of samples to be generated.

method	Which method to use for the sampling? The options are "conditional" and "Q". Here, "Q" is more stable but takes longer.
BC	Boundary conditions for degree 1 vertices. BC = 0 gives Neumann boundary conditions and BC = 1 gives stationary boundary conditions.

Details

Samples a Gaussian Whittle-Matérn field on a metric graph, either from the prior or conditionally on observations

$$y_i = u(t_i) + \sigma_e e_i$$

on the graph, where e_i are independent standard Gaussian variables. The parameters for the field can either be specified in terms of tau and kappa or practical correlation range and marginal standard deviation.

Value

Matrix or vector with the samples.

---

select.metric_graph_data

A version of `dplyr::select()` *function for datasets on metric graphs*

Description

Selects columns on metric graphs, while keeps the spatial positions.

Usage

```
## S3 method for class 'metric_graph_data'
select(.data, ...)
```

Arguments

.data	The data list or `tidyr::tibble` obtained from a metric graph object.
...	Additional parameters to be passed to `dplyr::select()`.

Value

A `tidyr::tibble` with the resulting selected columns.

selected_inv *Selected Inverse Calculation*

Description

Selected Inverse Calculation

Usage

```
selected_inv(Q)
```

Arguments

Q	A sparse matrix in dgCMatrix format

Value

A numeric vector containing the selected inverse

simulate.graph_lme *Simulation of models on metric graphs*

Description

The function samples a Gaussian random field based on a fitted model using `graph_lme()`.

Usage

```
## S3 method for class 'graph_lme'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  sample_latent = FALSE,
  posterior = FALSE,
  which_repl = NULL,
  ...
)
```

Arguments

object	A graph_lme object
nsim	The number of simulations.
seed	an object specifying if and how the random number generator should be initialized ('seeded').
sample_latent	If FALSE, samples for the response variable will be generated. If TRUE, samples for the latent model will be generated. The default is FALSE.
posterior	Should posterior samples be generated? If FALSE, samples will be computed based on the estimated prior distribution. The default is FALSE.
which_repl	Which replicates to generate the samples. If NULL samples will be generated for all replicates. Default is NULL.
...	Currently not used.

Value

A list containing elements samples, edge_number and distance_on_edge. Each of them is a list, whose indexes are the replicates, and in samples a matrix is given with nsim columns, each one being a sample. edge_number and distance_on_edges contain the respective edge numbers and distances on edge for each sampled element. The locations of the samples are the location of the data in which the model was fitted.

---

simulate.metric_graph *Simulate a Whittle-Matérn field on a metric graph*

Description

Draws unconditional (prior) samples of a Whittle-Matérn field on a metric graph using one of two exact algorithms from Section 6.6 of the paper.

Usage

```
## S3 method for class 'metric_graph'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  alpha = 1,
  method = c("direct", "kriging", "extended"),
  impl = c("cpp", "R"),
  kappa,
  tau,
  range,
  sigma,
  PtE = NULL,
  type = "manual",
```

```
  BC = 1,
  parallel = FALSE,
  n_cores = NULL,
  cluster = NULL,
  ...
)
```

Arguments

object	A metric_graph object.
nsim	Number of samples. Returns a vector (nsim = 1) or matrix (nsim > 1) of sampled values.
seed	Integer seed for set.seed() before the vertex draw. Pass NULL (default) to use the current RNG state.
alpha	Smoothness parameter (1 or 2).
method	Simulation algorithm: "direct" (Method A, $O(m^3)$), "kriging" (Method B, $O(m)$), or "extended" (add simulation locations as graph vertices, single sparse Cholesky draw).
impl	Implementation: "cpp" (default, Rcpp/Eigen) or "R" (pure R reference, for testing/comparison). Ignored for method = "extended".
kappa	Range parameter.
tau	Precision parameter.
range	Practical correlation range (alternative to kappa/tau).
sigma	Marginal standard deviation (alternative to kappa/tau).
PtE	Matrix with columns (edge_number, normalised_position). Required when type = "manual" (the default).
type	Location specification: "manual" (use PtE), "mesh" (mesh nodes), or "obs" (observation locations).
BC	Boundary condition for degree-1 vertices: 1 = stationary (default), 0 = Neumann.
parallel	Logical. Use a parallel cluster for the edge loop?
n_cores	Number of parallel workers. Ignored if cluster is supplied. Defaults to detectCores() - 1.
cluster	An already-registered doParallel cluster. If NULL and parallel = TRUE, a cluster is created and stopped automatically.
...	Ignored.

Details

Algorithm A ("direct"): interior values on each edge are drawn from the bridge conditional $\mathcal{N}(S_e b_e, \Sigma_e^*)$ via a dense Cholesky of the $m \times m$ bridge covariance. Cost: $O(m^3)$ per edge. Preferred when m is small (roughly $m \lesssim 100$).

Algorithm B ("kriging"): an unconditioned Matérn process is simulated on each edge by a linear Markov recursion, then the bridge boundary data are imposed by a kriging correction (Theorem 2 / Proposition 4). Cost: $O(m)$ per edge. Preferred for large m.

Both algorithms produce draws from exactly the same finite-dimensional distribution; they differ only in computational cost.

The two-step procedure is:

1. Draw the vertex (boundary) state from the vertex precision via a sparse Cholesky.
2. For each edge (independently given the vertex state), draw the interior values using the chosen algorithm.

When `parallel = TRUE` the edge loop is distributed over a `doParallel/foreach` cluster. Reproducibility requires passing a `seed` so that per-edge seeds are derived deterministically.

Value

Numeric vector (nsim = 1) or matrix with nsim columns of field values at the requested locations, ordered to match `PtE`.

See Also

`sample_spde()`

Examples

```
# Small square graph, alpha = 1
V <- rbind(c(0,0), c(1,0), c(1,1), c(0,1))
E <- rbind(c(1,2), c(2,3), c(3,4), c(4,1))
g <- metric_graph$new(V = V, E = E)
t_norm <- seq(0.1, 0.9, by = 0.2)
PtE    <- do.call(rbind, lapply(1:4, function(e) cbind(e, t_norm)))
u <- simulate(g, alpha = 1, method = "direct", kappa = 1, tau = 1, PtE = PtE)

# Method B on a larger graph, parallel
u2 <- simulate(g, alpha = 2, method = "kriging", range = 1, sigma = 1,
               PtE = PtE, parallel = TRUE, n_cores = 2)
```

`simulate_parallel` *Parallel simulation of a Whittle-Matérn field on a metric graph*

Description

Calls `simulate.metric_graph()` with `parallel = TRUE`. The edge loop is distributed over a `doParallel/foreach` cluster.

Usage

```
simulate_parallel(graph, ..., n_cores = NULL, cluster = NULL)
```

Arguments

graph	A metric_graph object.
...	Further arguments passed to simulate.metric_graph(), including alpha, method, kappa/tau or range/sigma, PtE, type, BC, nsim, and seed.
n_cores	Number of parallel workers.
cluster	An already-registered doParallel cluster.

Value

Numeric vector (nsim = 1) or matrix with nsim columns.

See Also

sample_spde()

Examples

```
V <- rbind(c(0,0), c(1,0), c(1,1), c(0,1))
E <- rbind(c(1,2), c(2,3), c(3,4), c(4,1))
g <- metric_graph$new(V = V, E = E)
t_norm <- seq(0.1, 0.9, by = 0.2)
PtE    <- do.call(rbind, lapply(1:4, function(e) cbind(e, t_norm)))
u <- simulate_parallel(g, alpha = 1, method = "kriging",
                       kappa = 1, tau = 1, PtE = PtE, n_cores = 2)
```

---

simulate_spacetime *space-time simulation based on implicit Euler discretization in time*

Description

Simulation with starting value u0

Usage

```
simulate_spacetime(graph, t, kappa, rho, gamma, alpha, beta, sigma, u0, BC = 0)
```

Arguments

graph	A metric_graph object.
t	Vector of time points.
kappa	Spatial range parameter.
rho	Drift parameter.
gamma	Temporal range parameter.
alpha	Smoothness parameter (integer) for spatial operator.

beta	Smoothness parameter (integer) for Q-Wiener process.
sigma	Variance parameter.
u0	Starting value.
BC	Which boundary condition to use (0,1). Here, 0 is no adjustment on the boundary and 1 results in making the boundary condition stationary.

Value

Precision matrix.

`spde_covariance` *Covariance function for Whittle-Matérn fields*

Description

Computes the covariance function for a Whittle-Matérn field.

Usage

```
spde_covariance(P, kappa, tau, range, sigma, alpha, graph, directional = F)
```

Arguments

P	Location (edge number and normalized location on the edge) for the location to evaluate the covariance function at.
kappa	Parameter kappa from the SPDE.
tau	Parameter tau from the SPDE.
range	Range parameter.
sigma	Standard deviation parameter.
alpha	Smoothness parameter (1 or 2).
graph	A `metric_graph` object.
directional	bool is the model a directional or not. directional only works for alpha=1

Details

Compute the covariance function $\rho(P, s_i)$ where P is the provided location and s_i are all locations in the mesh of the graph.

Value

Vector with the covariance function evaluate at the mesh locations.

spde_metric_graph_result *Metric graph SPDE result extraction from 'INLA' estimation results*

Description

Extract field and parameter values and distributions for a metric graph spde effect from an 'INLA' result object.

Usage

```
spde_metric_graph_result(
  inla,
  name,
  metric_graph_spde,
  compute.summary = TRUE,
  n_samples = 5000,
  n_density = 1024
)
```

Arguments

inla	An 'INLA' object obtained from a call to inla().
name	A character string with the name of the 'rSPDE' effect in the model.
metric_graph_spde	The inla_metric_graph_spde object used for the random effect in the model.
compute.summary	Should the summary be computed?
n_samples	The number of samples to be used if parameterization is matern.
n_density	The number of equally spaced points to estimate the density.

Value

If the model was fitted with matern parameterization (the default), it returns a list containing:

marginals.range	Marginal densities for the range parameter.
marginals.log.range	Marginal densities for log(range).
marginals.sigma	Marginal densities for std. deviation.
marginals.log.sigma	Marginal densities for log(std. deviation).
marginals.values	Marginal densities for the field values.

summary.log.range
: Summary statistics for log(range).

summary.log.sigma
: Summary statistics for log(std. deviation).

summary.values
: Summary statistics for the field values.

If compute.summary is TRUE, then the list will also contain

summary.kappa
: Summary statistics for kappa.

summary.tau
: Summary statistics for tau.

If the model was fitted with the spde parameterization, it returns a list containing:

marginals.kappa
: Marginal densities for kappa.

marginals.log.kappa
: Marginal densities for log(kappa).

marginals.log.tau
: Marginal densities for log(tau).

marginals.tau
: Marginal densities for tau.

marginals.values
: Marginal densities for the field values.

summary.log.kappa
: Summary statistics for log(kappa).

summary.log.tau
: Summary statistics for log(tau).

summary.values
: Summary statistics for the field values.

If compute.summary is TRUE, then the list will also contain

summary.kappa
: Summary statistics for kappa.

summary.tau
: Summary statistics for tau.

---

spde_precision *Precision matrix for Whittle-Matérn fields*

Description

Computes the precision matrix for all vertices for a Whittle-Matérn field.

Usage

```
spde_precision(kappa, tau, alpha, graph, BC = 1, build = TRUE)
```

Arguments

kappa	Range parameter.
tau	Precision parameter.
alpha	Smoothness parameter (1 or 2).
graph	A metric_graph object.
BC	Set boundary conditions for degree=1 vertices. BC =0 gives Neumann boundary conditions and BC=1 gives stationary boundary conditions.
build	If TRUE, the precision matrix is returned. Otherwise a list list(i,j,x, nv) is returned.

Value

Precision matrix or list.

---

spde_variance *Variancefor Whittle-Matérn fields*

Description

Computes the variance function for a Whittle-Matérn field. Warning is not feasible for large graph due to matrix inversion

Usage

```
spde_variance(
  kappa,
  tau,
  range,
  sigma,
  alpha,
  graph,
  BC = 1,
  include_vertices = FALSE,
  directional = F
)
```

Arguments

kappa	Parameter kappa from the SPDE.
tau	Parameter tau from the SPDE.
range	Range parameter.
sigma	Standard deviation parameter.
alpha	Smoothness parameter (1 or 2).
graph	A metric_graph object.

BC	boundary conditions
include_vertices	Should the variance at the vertices locations be included in the returned vector?
directional	bool is the model a directional or not. directional only works for alpha=1

Details

Compute the variance $\rho(s_i, s_i)$ where s_i are all locations in the mesh of the graph.

Value

Vector with the variance function evaluate at the mesh locations.

`stlpp.to.graph` *Convert an* `stlpp` *object to a metric graph object*

Description

This function converts an `stlpp` object (from the `stlnpp` package) into a metric graph object.

Usage

```
stlpp.to.graph(stlpp.obj, ...)
```

Arguments

stlpp.obj	An `stlpp` object to be converted.
...	Additional arguments to be passed to the `metric_graph` constructor.

Value

A metric graph object

`summarise.metric_graph_data` *A version of* `dplyr::summarise()` *function for datasets on metric graphs*

Description

Creates summaries, while keeps the spatial positions.

Usage

```
## S3 method for class 'metric_graph_data'
summarise(.data, ..., .include_graph_groups = FALSE, .groups = NULL)
```

Arguments

.data	The data list or tidyr::tibble obtained from a metric graph object.
...	Additional parameters to be passed to dplyr::summarise().
.include_graph_groups	Should the internal graph groups be included in the grouping variables? The default is FALSE. This means that, when summarising, the data will be grouped by the internal group variable together with the spatial locations.
.groups	A vector of strings containing the names of the columns to be additionally grouped, when computing the summaries. The default is NULL.

Value

A tidyr::tibble with the resulting selected columns.

---

summary.graph_lme *Summary Method for* graph_lme *Objects*

Description

Function providing a summary of results related to metric graph mixed effects regression models.

Usage

```
## S3 method for class 'graph_lme'
summary(object, all_times = FALSE, ...)
```

Arguments

object	an object of class graph_lme containing results from the fitted model.
all_times	Show all computed times.
...	not used.

Value

An object of class summary_graph_lme containing information about a *graph_lme* object.

summary.metric_graph *Summary Method for* metric_graph *Objects*

Description

Function providing a summary of several informations/characteristics of a metric graph object.

Usage

```
## S3 method for class 'metric_graph'
summary(
  object,
  messages = FALSE,
  compute_characteristics = NULL,
  check_euclidean = NULL,
  check_distance_consistency = NULL,
  ...
)
```

Arguments

object	an object of class metric_graph.
messages	Should message explaining how to build the results be given for missing quantities?
compute_characteristics	Should the characteristics of the graph be computed? If NULL it will be determined based on the size of the graph.
check_euclidean	Check if the graph has Euclidean edges? If NULL it will be determined based on the size of the graph.
check_distance_consistency	Check the distance consistency assumption?#' If NULL it will be determined based on the size of the graph.
...	not used.

Value

An object of class summary_graph_lme containing information about a *metric_graph* object.

summary.metric_graph_spde_result *Summary for posteriors of field parameters for an* inla_rspde *model from a* rspde.result *object*

Description

Summary for posteriors of 'rSPDE' field parameters in their original scales.

Usage

```
## S3 method for class 'metric_graph_spde_result'
summary(object, digits = 6, ...)
```

Arguments

object	A rspde.result object.
digits	Integer, used for number formatting with signif()
...	Currently not used.

Value

A data.frame containing the summary.

update_graph *Update an older version metric graph to the current package version*

Description

This function creates a new metric_graph object from an existing one, preserving all data, edge weights, mesh, distances, and other components while ensuring compatibility with the current package version. The new graph is created with minimal changes by disabling merges and other transformations.

Usage

```
update_graph(old_graph, verbose = TRUE)
```

Arguments

old_graph	A metric_graph object from an older version of the package
verbose	Logical. Print progress messages. Default is TRUE.

Details

This function is useful when loading metric graphs created with older versions of the MetricGraph package. It ensures that all components (vertices, edges, data, mesh, distances, etc.) are preserved while updating the internal structure to be compatible with the current package version.

The function works by:

- Extracting all components from the old graph
- Creating a new graph with settings that minimize structural changes
- Restoring all computed components (mesh, distances, etc.)
- Re-adding observation data if present

Value

A new metric_graph object compatible with the current package version

Examples

```
## Not run:
# Load an old graph object
old_graph <- readRDS("old_graph.rds")

# Update to current version
new_graph <- update_graph(old_graph)

# Or use the method directly
new_graph <- old_graph$update_graph()

## End(Not run)
```

---

weighted_stiffness *Assemble a weighted stiffness matrix on a graph mesh*

Description

Uses a piecewise linear positive coefficient on each mesh edge. The coefficient integral on an edge is its length times the mean of its two endpoint values.

Usage

```
weighted_stiffness(edges, lengths, weights)
```

Arguments

edges	Two-column integer matrix of mesh edge endpoints.
lengths	Positive mesh edge lengths, in the same order as rows of edges.
weights	Positive coefficient values at mesh nodes.

Value

A symmetric sparse positive-semidefinite stiffness matrix.

Index

∗ **datasets**
pems, 72
pems_repl, 73

augment *(*augment.graph_lme*)*, 4
augment.graph_lme, 4, *17*

bru_get_mapper.inla_metric_graph_spde *(*bru_mapper.inla_metric_graph_spde*)*, 7
bru_mapper.inla_metric_graph_spde, 7

cross_validation, 7

directional_ou_covariance, 9
directional_ou_covariance(), *11, 12*
directional_ou_variance, 11
directional_ou_variance(), *11*
drop_na *(*drop_na.metric_graph_data*)*, 12
drop_na.metric_graph_data, 12

exp_covariance, 13

fetch_osm, 14
fetch_osm(), *70, 71*
filter *(*filter.metric_graph_data*)*, 15
filter.metric_graph_data, 15

gg_df *(*gg_df.metric_graph_spde_result*)*, 16
gg_df.metric_graph_spde_result, 16
glance *(*glance.graph_lme*)*, 17
glance.graph_lme, *6*, 17
graph_bru_process_data, 18
graph_data_spde, 18
graph_lgcp_sim, 20, *32*, *79*
graph_lme, 22
graph_spde, 24, *32*, *79*
graph_spde_basis, 28
graph_spde_make_A, 28
graph_starting_values, 29

ibm_jacobian.bru_mapper_inla_metric_graph_spde *(*bru_mapper.inla_metric_graph_spde*)*, 7
ibm_n.bru_mapper_inla_metric_graph_spde *(*bru_mapper.inla_metric_graph_spde*)*, 7
ibm_values.bru_mapper_inla_metric_graph_spde *(*bru_mapper.inla_metric_graph_spde*)*, 7

lgcp_graph, *21*, 30, *79*
linnet.to.graph, 33
logo_lines, 34

make_Q_euler, 34
make_Q_spacetime, 35
match_mesh_data, 35
metric_graph, 36, *70, 71*
metric_graph_from_osm, 70
metric_graph_from_osm(), *15*
MetricGraph *(*MetricGraph-package*)*, 3
MetricGraph-package, 3
mutate *(*mutate.metric_graph_data*)*, 72
mutate.metric_graph_data, 72

osmdata::add_osm_feature(), *14*, *71*
osmdata::opq(), *14*, *70, 71*
osmdata::osmdata_sf(), *14, 15*

pems, 72
pems_repl, 73
plot.graph_bru_pred, 74
plot.graph_bru_proc_pred, 74
posterior_crossvalidation *(*posterior_crossvalidation.graph_lme*)*, 75
posterior_crossvalidation.graph_lme, 75
posterior_crossvalidation_loo, 76
precompute_lgcp_graph, *21*, *32*, 77

predict.graph_lme, 81
predict.inla_metric_graph_spde, 83
predict.rspde_metric_graph, 85
process_rspde_predictions, 87
psp.to.graph, 87

R6Class, *36*
rSPDE::cross_validation(), *7*
rSPDE::posterior_crossvalidation(), *75*

sample_spde, 88
sample_spde(), *93, 94*
select *(*select.metric_graph_data*)*, 89
select.metric_graph_data, 89
selected_inv, 90
simulate.graph_lme, 90
simulate.metric_graph, 91
simulate_parallel, 93
simulate_spacetime, 94
spde_covariance, 95
spde_metric_graph_result, *32*, *79*, 96
spde_precision, 97
spde_variance, 98
stlpp.to.graph, 99
summarise
*(*summarise.metric_graph_data*)*, 99
summarise.metric_graph_data, 99
summary.graph_lme, 100
summary.metric_graph, 101
summary.metric_graph_spde_result, 102

tidyr::tibble(), *6*, *17*

update_graph, 102

weighted_stiffness, 103