Package 'MetricGraph'

March 2, 2025

Type Package

Title Random Fields on Metric Graphs

Version 1.4.1

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, igraph, sf, rSPDE (>= 2.3.3), Matrix, methods, Rcpp (>= 1.0.5), R6, lifecycle, sp, dplyr, tidyr, magrittr, broom, zoo, ggnewscale

Suggests knitr, testthat, INLA (>= 22.12.14), inlabru, osmdata, sn, plotly, parallel, optimParallel, numDeriv, SSN2, cowplot, leaflet, mapview, viridis, fmesher

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.

Contents

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:

- 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

```
bru_get_mapper.inla_metric_graph_spde(model, ...)

ibm_n.bru_mapper_inla_metric_graph_spde(mapper, ...)

ibm_values.bru_mapper_inla_metric_graph_spde(mapper, ...)

ibm_jacobian.bru_mapper_inla_metric_graph_spde(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

---

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.

---

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()`](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()`](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_components   *Connected components of metric graph*

Description

Class representing connected components of a metric graph.

Details

A list of `metric_graph` objects (representing the different connected components in the full graph) 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 graph components.

Public fields

`graphs` List of the graphs representing the connected components.

`n` The number of graphs.

`sizes` Number of vertices for each of the graphs.

`lengths` Total edge lengths for each of the graphs. Create metric graphs for connected components

Methods

Public methods:

- [graph_components$new()](#)
- [graph_components$get_largest()](#)
- [graph_components$plot()](#)
- [graph_components$clone()](#)

Method `new()`:

Usage:

```
graph_components$new(
  edges = NULL,
  V = NULL,
  E = NULL,
  by_length = TRUE,
  edge_weights = NULL,
  ...,
  lines = deprecated()
)
```

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 `SpatialLinesDataFrame` or `SpatialLines` (from sp package) or `MULTILINESTRING` (from `sf` package).

`V` n x 2 matrix with Euclidean coordinates of the n vertices.

`E` m x 2 matrix where each row represents an edge.

`by_length` Sort the components by total edge length? If `FALSE`, the components are sorted by the number of vertices.

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

... Additional arguments used when specifying the graphs

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

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 is vertex_unit. Observe that if vertex_unit is NULL, length_unit can only be NULL. If vertex_unit is 'degree', then the default value for length_unit is 'km'.

longlat If TRUE, then it is assumed that the coordinates are given. in Longitude/Latitude and that distances should be computed in meters. It takes precedence over vertex_unit and length_unit, and is equivalent to vertex_unit = 'degree' and length_unit = 'm'.

tolerance Vertices that are closer than this number are merged when constructing the graph (default = 1e-10). If longlat = TRUE, the tolerance is given in km.

Returns: A graph_components object.

Method get_largest(): Returns the largest component in the graph.

Usage:
graph_components$get_largest()

Returns: A metric_graph object.

Method plot(): Plots all components.

Usage:
graph_components$plot(edge_colors = NULL, vertex_colors = NULL, ...)

Arguments:

edge_colors A 3 x nc matrix with RGB values for the edge colors to be used when plotting each graph.

vertex_colors A 3 x nc matrix with RGB values for the edge colors to be used when plotting each graph.

... Additional arguments for plotting the individual graphs.

Returns: A ggplot object.

Method clone(): The objects of this class are cloneable with this method.

Usage:
graph_components$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

Examples

```
library(sp)
edge1 <- rbind(c(0, 0), c(1, 0))
edge2 <- rbind(c(1, 0), c(2, 0))
edge3 <- rbind(c(1, 1), c(2, 1))
edges <- list(edge1, edge2, edge3)

graphs <- graph_components$new(edges)
graphs$plot()
```

---

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	*Simulation of log-Gaussian Cox processes driven by Whittle-Matérn fields on metric graphs*

Description

Simulation of log-Gaussian Cox processes driven by Whittle-Matérn fields on metric graphs

Usage

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

Arguments

n	Number of samples.
intercept	Mean value of the Gaussian process.
sigma	Parameter for marginal standard deviations.
range	Parameter for practical correlation range.
alpha	Smoothness parameter (1 or 2).
graph	A `metric_graph` object.

Value

List with Gaussian process sample and simulated points.

---

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 $\alpha = 1$ 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', 'iso-Exp', '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,
  directional = FALSE,
  stationary_endpoints = "all",
  parameterization = c("matern", "spde"),
  start_range = NULL,
  prior_range = NULL,
  start_kappa = NULL,
  prior_kappa = NULL,
  start_sigma = NULL,
  prior_sigma = NULL,
  start_tau = NULL,
  prior_tau = NULL,
  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.
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.
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.
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.

start_tau   Starting value for tau.

prior_tau   a `list` containing the elements `meanlog` and `sdlog`, that is, the mean and stan-
                     dard deviation of tau on the log scale.

factor_start_range

                     Factor to multiply the max/min dimension of the bounding box to obtain a start-
                     ing 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", it will
                     check if the shared lib exists locally, in which case it will use it. Otherwise
                     it will use 'INLA's shared library. If 'INLA', it will use the shared lib from
                     'INLA's installation. If 'rSPDE', then it will use the local installation of the
                     rSPDE package (does not work if your installation is from CRAN). 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)`

---

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.

---

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$buildC()`
- `metric_graph$build_mesh()`
- `metric_graph$compute_fem()`
- `metric_graph$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$get_initial_graph()`
- `metric_graph$coordinates()`
- `metric_graph$clone()`

Method 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()
)
```

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 op-
tions 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.

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.

Method 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.

Method `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.

Method `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

Method `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.

Method `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`.

Method `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()`.

Method `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.

Method `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.

Method `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.

Method `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.

Method `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.

Method `print()`: Prints various characteristics of the graph

Usage:
```
metric_graph$print()
```

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

Method `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.

Method `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.

Method `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.

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

Usage:

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

Arguments:

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

`obs` Should the geodesic 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.

`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.

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.

Method `compute_geodist_PtE()`: Computes shortest path distances between the vertices in the graph.

Usage:
```
metric_graph$compute_geodist_PtE(
  PtE,
  normalized = TRUE,
  include_vertices = TRUE,
  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 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 1.

Returns: A matrix containing the geodesic distances.

Method `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.

Method `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.

Method 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.

Method 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.

Method 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.

Method 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.

Method 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.

Method 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.

Method 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.

Method 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.

Method get_PtE(): Gets PtE from the data.

Usage:
metric_graph$get_PtE()

Arguments:

group For which group, should the PtE be returned? NULL means that all PtEs available will be returned.

include_group Should the group be included as a column? If TRUE, the PtEs for each group will be concatenated, otherwise a single matrix containing the unique PtEs will be returned.

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.

Method 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).

Method 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.

Method 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.

Method 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).

Method `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.

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

Usage:

```
metric_graph$clear_observations()
```

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

Method `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 argu-
    ment 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.

Method 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 argu-
        ment 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 met-
        ric 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 ob-
        servations 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 op-
tions 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.

Method 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.

Method 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.

Method 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.

Method 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.

Method 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.

Method 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.

Method 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.

Method `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.

Method `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.

Method `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.

Method `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.

Method `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.

Method `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.

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

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

Method `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.

Method 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.

Method 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.

Method 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.

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

Usage:
```
metric_graph$VtEfirst()
```

Returns: A nV x 2 matrix the first element of the ith row is the edge number corresponding to the ith 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.

Method `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(),
  ...
)
```

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.

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

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

Method plot_connections(): Plots the connections in the graph

Usage:
```
metric_graph$plot_connections()
```

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

Method 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.

Method plot_function(): Plots continuous function on the graph.

Usage:
```
metric_graph$plot_function(
  data = NULL,
  newdata = NULL,
  group = 1,
  X = NULL,
  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(),
    ...
)
```

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.

X A vector with values for the function evaluated at the mesh in the graph

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.

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

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

Method `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.

Method 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.

Method get_initial_graph(): Returns a copy of the initial metric graph.

Usage:

metric_graph$get_initial_graph()

Returns: A metric_graph object.

Method 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.

Method 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()
```

---

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

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

Description

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

Usage

```
posterior_crossvalidation(object, factor = 1, tibble = TRUE)
```

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()

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.

---

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,
  ...,
  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.

...   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 pre-
diction 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 "1:1"
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 "1:1"
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_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 re-
               turned.

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.

Index