

Package ‘inlabru’

March 30, 2022

Type Package

Title Bayesian Latent Gaussian Modelling using INLA and Extensions

Version 2.5.2

URL <http://www.inlabru.org>, <https://inlabru-org.github.io/inlabru/>

BugReports <https://github.com/inlabru-org/inlabru/issues>

Description Facilitates spatial and general latent Gaussian modeling using integrated nested Laplace approximation via the INLA package (<<https://www.r-inla.org>>). Additionally, extends the GAM-like model class to more general nonlinear predictor expressions, and implements a log Gaussian Cox process likelihood for modeling univariate and spatial point processes based on ecological survey data. Model components are specified with general inputs and mapping methods to the latent variables, and the predictors are specified via general R expressions, with separate expressions for each observation likelihood model in multi-likelihood models. A prediction method based on fast Monte Carlo sampling allows posterior prediction of general expressions of the latent variables. Ecology-focused introduction in Bachl, Lindgren, Borchers, and Illian (2019) <[doi:10.1111/2041-210X.13168](https://doi.org/10.1111/2041-210X.13168)>.

License GPL (>= 2)

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

RoxygenNote 7.1.2

Encoding UTF-8

Depends methods, R (>= 3.5), sp (>= 1.4-5), stats

Imports MatrixModels, magrittr, Matrix, plyr, rgdal (>= 1.5.8), rgeos, rlang, utils, withr

Suggests covr, dplyr, ggmap, ggplot2, ggpolypath, graphics, INLA (>= 21.08.31), knitr, maptools, mgcv, patchwork, raster, RColorBrewer, rgl, rmarkdown, shiny, sn, spatstat.geom, spatstat.core, spatstat.data, spatstat (>= 2.0-0), sphereplot, testthat

Config/testthat/parallel true

Config/testthat/edition 3

Collate '0_inlabru_envir.R' 'bru.covariate.R' 'bru.gof.R'
 'bru.inference.R' 'bru.integration.R' 'bru.spatial.R'
 'covariate.R' 'data.Poisson1_1D.R' 'data.Poisson2_1D.R'
 'data.Poisson3_1D.R' 'data.gorillas.R' 'data.mexdolphin.R'
 'data.mrsea.R' 'data.seals.R' 'data.shrimp.R'
 'data.toygroups.R' 'deltaIC.R' 'deprecated.R' 'dsdata.R'
 'dsmdata.R' 'dsmdata.tools.R' 'effect.R' 'environment.R'
 'fmesher_crs.R' 'fmesher_utils.R' 'ggplot.R' 'inla.R'
 'inlabru.R' 'integration.R' 'local_testthat.R' 'mesh.R'
 'model.R' 'nlinla.R' 'plotsample.R' 'prediction.R' 'rgl.R'
 'sampling.R' 'shapefile.R' 'spatstat.R' 'spde.R' 'stack.R'
 'track_plotting.R' 'transformation.R' 'utils.R'

VignetteBuilder knitr

BuildVignettes true

NeedsCompilation no

Author Finn Lindgren [aut, cre, cph] (<<https://orcid.org/0000-0002-5833-2011>>,
 Finn Lindgren continued development of the main code),
 Fabian E. Bachl [aut, cph] (Fabian Bachl wrote the main code),
 David L. Borchers [ctb, dtc, cph] (David Borchers wrote code for
 Gorilla data import and sampling, multiplot tool),
 Daniel Simpson [ctb, cph] (Daniel Simpson wrote the basic LGCP sampling
 method),
 Lindesay Scott-Howard [ctb, dtc, cph] (Lindesay Scott-Howard provided
 MRSea data import code),
 Seaton Andy [ctb] (Andy Seaton provided testing and bugfixes),
 Suen Man Ho [ctb, cph] (Man Ho Suen contributed features for aggregated
 responses)

Maintainer Finn Lindgren <finn.lindgren@gmail.com>

Repository CRAN

Date/Publication 2022-03-30 16:20:05 UTC

R topics documented:

bincount	4
bru	6
bru_call_options	8
bru_compute_linearisation	8
bru_fill_missing	10
bru_forward_transformation	11
bru_log_reset	12
bru_make_stack	14
bru_mapper	15
bru_mapper_methods	17
bru_options	22
bru_safe_inla	25
bru_standardise_names	26

component	26
component_eval	30
component_list	31
cprod	33
deltaIC	34
devel.cvmeasure	35
evaluate_A	38
evaluate_index	38
expand_labels	39
fm_CRS	39
fm_has_PROJ6	41
fm_spTransform	42
fm_sp_get_crs	43
fm_wkt_as_wkt_tree	44
fm_wkt_is_geocent	45
generate	46
gg	49
gg.data.frame	50
gg.inla.mesh	52
gg.inla.mesh.1d	54
gg.matrix	56
gg.prediction	57
gg.RasterLayer	59
gg.SpatialGridDataFrame	60
gg.SpatialLines	62
gg.SpatialPixels	63
gg.SpatialPixelsDataFrame	64
gg.SpatialPoints	66
gg.SpatialPolygons	68
globe	70
gplot	71
gplot.inla.mesh	72
gplot.SpatialLines	73
gplot.SpatialPoints	74
gm	75
gmap	76
gorillas	77
inla.stack.mjoin	79
inlabru	80
int	80
ipoints	81
lgcp	84
like	86
mexdolphins	89
mrsea	91
multiplot	92
pixels	93
plot.bru	94

plot.prediction	95
plotsample	97
point2count	98
Poisson1_1D	99
Poisson2_1D	100
Poisson3_1D	101
predict.bru	103
row_kron	106
sample.lgcp	107
seals	109
shrimp	110
sline	111
spatial.to.ppp	112
spde.posterior	113
spoly	115
stransform	116
summary.bru	117
summary.bru_info	118
summary.bru_options	119
summary_bru	120
toysgroups	121
vertices	122
vertices.inla.mesh	123

Index**124**

bincount

*1D LGCP bin count simulation and comparison with data***Description**

A common procedure of analyzing the distribution of 1D points is to chose a binning and plot the data's histogram with respect to this binning. This function compares the counts that the histogram calculates to simulations from a 1D log Gaussian Cox process conditioned on the number of data samples. For each bin this results in a median number of counts as well as a confidence interval. If the LGCP is a plausible model for the observed points then most of the histogram counts (number of points within a bin) should be within the confidence intervals. Note that a proper comparison is a multiple testing problem which the function does not solve for you.

Usage

```
bincount(
  result,
  predictor,
  observations,
  breaks,
  nint = 20,
  probs = c(0.025, 0.5, 0.975),
```

```
    ...
  )
```

Arguments

result	A result object from a <code>bru()</code> or <code>lgcp()</code> call
predictor	A formula describing the prediction of a 1D LGCP via <code>predict()</code> .
observations	A vector of observed values
breaks	A vector of bin boundaries
nint	Number of integration points per bin. Increase this if the bins are wide and
probs	numeric vector of probabilities with values in [0,1]
...	arguments passed on to <code>predict.bru()</code>

Value

An `data.frame` with a `ggplot` attribute `ggp`

Examples

```
## Not run:
if (require(ggplot2)) {
  # Load a point pattern
  data(Poisson2_1D)

  # Take a look at the point (and frequency) data

  ggplot(pts2) +
    geom_histogram(aes(x = x), binwidth = 55 / 20, boundary = 0, fill = NA, color = "black") +
    geom_point(aes(x), y = 0, pch = "|", cex = 4) +
    coord_fixed(ratio = 1)

  # Fit an LGCP model
  x <- seq(0, 55, length = 50)
  mesh1D <- inla.mesh.1d(x, boundary = "free")
  mdl <- x ~ spde1D(x, model = inla.spde2.matern(mesh1D)) + Intercept(1)
  fit.spde <- lgcp(mdl, pts2, domain = list(x = c(0, 55)))

  # Calculate bin statistics
  bc <- bincount(
    result = fit.spde,
    observations = pts2,
    breaks = seq(0, max(pts2), length = 12),
    predictor = x ~ exp(spde1D + Intercept)
  )

  # Plot them!
  attributes(bc)$ggp
}
```

```
## End(Not run)
```

bru

Convenient model fitting using (iterated) INLA

Description

This method is a wrapper for `INLA::inla` and provides multiple enhancements.

- Easy usage of spatial covariates and automatic construction of inla projection matrices for (spatial) SPDE models. This feature is accessible via the `components` parameter. Practical examples on how to use spatial data by means of the `components` parameter can also be found by looking at the `lgcp` function's documentation.
- Constructing multiple likelihoods is straight forward. See `like` for more information on how to provide additional likelihoods to `bru` using the `...` parameter list.
- Support for non-linear predictors. See example below.
- Log Gaussian Cox process (LGCP) inference is available by using the `cp` family or (even easier) by using the `lgcp` function.

Usage

```
bru(components = ~Intercept(1), ..., options = list(), .envir = parent.frame())
```

```
bru_rerun(result, options = list())
```

Arguments

<code>components</code>	A formula-like specification of latent components. Also used to define a default linear additive predictor. See <code>component()</code> for details.
<code>...</code>	Likelihoods, each constructed by a calling <code>like()</code> , or named parameters that can be passed to a single <code>like()</code> call. Note that all the arguments will be evaluated before calling <code>like()</code> in order to detect if they are <code>like</code> objects. This means that special arguments that need to be evaluated in the context of <code>response_data</code> or <code>data</code> (such as <code>Ntrials</code>) may will only work that way in direct calls to <code>like()</code> .
<code>options</code>	A <code>bru_options</code> options object or a list of options passed on to <code>bru_options()</code>
<code>.envir</code>	Environment for component evaluation (for when a non-formula specification is used)
<code>result</code>	A previous estimation object of class <code>bru</code>

Details

- `bru_rerun` Continue the optimisation from a previously computed estimate.

Value

bru returns an object of class "bru". A bru object inherits from INLA::inla (see the inla documentation for its properties) and adds additional information stored in the bru_info field.

Author(s)

Fabian E. Bachl <bachlfab@gmail.com>

Examples

```
if (bru_safe_inla(multicore = FALSE)) {  
  
  # Simulate some covariates x and observations y  
  input.df <- data.frame(x = cos(1:10))  
  input.df <- within(input.df, y <- 5 + 2 * x + rnorm(10, mean = 0, sd = 0.1))  
  
  # Fit a Gaussian likelihood model  
  fit <- bru(y ~ x + Intercept, family = "gaussian", data = input.df)  
  
  # Obtain summary  
  fit$summary.fixed  
}  
  
if (bru_safe_inla(multicore = FALSE)) {  
  
  # Alternatively, we can use the like() function to construct the likelihood:  
  
  lik <- like(family = "gaussian", formula = y ~ x + Intercept, data = input.df)  
  fit <- bru(~ x + Intercept(1), lik)  
  fit$summary.fixed  
}  
  
# An important addition to the INLA methodology is bru's ability to use  
# non-linear predictors. Such a predictor can be formulated via like()'s  
# \code{formula} parameter. The z(1) notation is needed to ensure that  
# the z component should be interpreted as single latent variable and not  
# a covariate:  
  
if (bru_safe_inla(multicore = FALSE)) {  
  z <- 2  
  input.df <- within(input.df, y <- 5 + exp(z) * x + rnorm(10, mean = 0, sd = 0.1))  
  lik <- like(  
    family = "gaussian", data = input.df,  
    formula = y ~ exp(z) * x + Intercept  
  )  
  fit <- bru(~ z(1) + Intercept(1), lik)  
  
  # Check the result (z posterior should be around 2)  
  fit$summary.fixed  
}
```

bru_call_options *Additional bru options*

Description

Construct a bru_options object including the default and global options, and converting deprecated option names.

Usage

```
bru_call_options(...)
```

Arguments

... Options passed on to [as.bru_options\(\)](#)

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

Examples

```
opts <- bru_call_options()

# Print them:
opts
```

bru_compute_linearisation
Compute inlabru model linearisation information

Description

Compute inlabru model linearisation information

Usage

```
bru_compute_linearisation(...)

## S3 method for class 'component'
bru_compute_linearisation(
  cmp,
  model,
  lhood_expr,
  data,
  state,
  A,
  effects,
  pred0,
  allow_latent,
  allow_combine,
  eps,
  ...
)

## S3 method for class 'bru_like'
bru_compute_linearisation(lhood, model, data, state, A, eps, ...)

## S3 method for class 'bru_like_list'
bru_compute_linearisation(lhoods, model, state, A, eps = 1e-05, ...)

## S3 method for class 'bru_model'
bru_compute_linearisation(model, lhoods, state, A, ...)
```

Arguments

...	Parameters passed on to other methods
cmp	A bru_component object
model	A bru_model object
lhood_expr	A predictor expression
data	Input data
state	The state information, as a list of named vectors
A	A-matrix information: <ul style="list-style-type: none"> • For bru_component: Precomputed A-matrix for the component • For bru_like: A list of named A-matrices for the components in the likelihood for the component • For bru_like_list: A list, where each element is a list of named A-matrices.
effects	<ul style="list-style-type: none"> • For bru_component: Precomputed effect list for all components involved in the likelihood expression
pred0	Precomputed predictor for the given state

allow_latent	logical. If TRUE, the latent state of each component is directly available to the predictor expression, with a <code>_latent</code> suffix.
allow_combine	logical; If TRUE, the predictor expression may involve several rows of the input data to influence the same row.
eps	The finite difference step size
lhood	A <code>bru_like</code> object
lhoods	A <code>bru_like_list</code> object

`bru_fill_missing` *Fill in missing values in Spatial grids*

Description

Computes nearest-available-value imputation for missing values in space

Usage

```
bru_fill_missing(
  data,
  where,
  values,
  layer = NULL,
  selector = NULL,
  batch_size = 500
)
```

Arguments

<code>data</code>	A <code>SpatialPointsDataFrame</code> , <code>SpatialPixelsDataFrame</code> , or a <code>SpatialGridDataFrame</code> containing data to use for filling
<code>where</code>	A, matrix, <code>data.frame</code> , or <code>SpatialPoints</code> or <code>SpatialPointsDataFrame</code> , containing the locations of the evaluated values
<code>values</code>	A vector of values to be filled in where <code>is.na(values)</code> is TRUE
<code>layer, selector</code>	Specifies what data column or columns from which to extract data, see component() for details.
<code>batch_size</code>	Size of nearest-neighbour calculation blocks, to limit the memory and computational complexity.

Value

An infilled vector of values

Examples

```
## Not run:
if (bru_safe_inla()) {
  points <-
    sp::SpatialPointsDataFrame(
      matrix(1:6, 3, 2),
      data = data.frame(val = c(NA, NA, NA))
    )
  input_coord <- expand.grid(x = 0:7, y = 0:7)
  input <-
    sp::SpatialPixelsDataFrame(
      input_coord,
      data = data.frame(val = as.vector(input_coord$y))
    )
  points$val <- bru_fill_missing(input, points, points$val)
  print(points)

  # To fill in missing values in a grid:
  print(input$val[c(3, 30)])
  input$val[c(3, 30)] <- NA # Introduce missing values
  input$val <- bru_fill_missing(input, input, input$val)
  print(input$val[c(3, 30)])
}

## End(Not run)
```

bru_forward_transformation

Transformation tools

Description

Tools for transforming between $N(0,1)$ variables and other distributions in predictor expressions

Usage

```
bru_forward_transformation(qfun, x, ..., tail.split. = 0)
```

```
bru_inverse_transformation(pfun, x, ..., tail.split. = NULL)
```

Arguments

qfun	A quantile function object, such as qexp
x	Values to be transformed
...	Distribution parameters passed on to the qfun and pfun functions

`tail.split.` For x-values larger than `tail.split.`, upper quantile calculations are used internally, and for smaller values lower quantile calculations are used. This can avoid lack of accuracy in the distribution tails. If NULL, forward calculations split at 0, and inverse calculations use lower tails only, potentially losing accuracy in the upper tails.

`pfun` A CDF function object, such as `pexp`

Value

- For `bru_forward_transformation`, a numeric vector
- For `bru_inverse_transformation`, a numeric vector

Examples

```
## Not run:
if (interactive()) {
  # EXAMPLE1
}
## End(Not run)
```

`bru_log_reset` *inlabru log message methods*

Description

Resets the `inlabru log` object
 Retrieve, add, and/or print log messages

Usage

```
bru_log_reset()

bru_log_get(pretty = FALSE)

bru_log_message(
  ...,
  domain = NULL,
  appendLF = TRUE,
  verbosity = 1,
  allow_verbose = TRUE,
  verbose = NULL,
  verbose_store = NULL
)

bru_log(txt, verbose = NULL)

bru_log_active(activation = NULL)
```

Arguments

pretty	logical; If TRUE, return a single string with the log messages separated and terminated by line feeds, suitable for <code>cat(...)</code> . If FALSE, return the raw log as a vector of strings, suitable for <code>cat(..., sep = "\n")</code> . Default: FALSE
...	Zero or more objects passed on to <code>base::makeMessage()</code>
domain	Domain for translations, passed on to <code>base::makeMessage()</code>
appendLF	logical; whether to add a newline to the message. Only used for verbose output.
verbosity	numeric value describing the verbosity level of the message
allow_verbose	Whether to allow verbose output. Must be set to FALSE until the options object has been initialised.
verbose	logical; if TRUE, print the log message on screen with <code>message(txt)</code> . Default: <code>bru_options_get("bru_verbose")</code>
verbose_store	Same as verbose, but controlling what messages are stored in the global log object. Can be controlled via the <code>bru_verbose_store</code> with <code>bru_options_set()</code> .
txt	character; log message.
activation	logical; whether to activate (TRUE) or deactivate (FALSE) the inlabru logging system. Default: NULL, to keep the current activation state

Details

`bru_log_reset()` clears the log contents.

- `bru_log_message` DETAILS

The log message is stored if the log is active, see `bru_log_active()`

Value

`bru_log_get` RETURN_VALUE

- `bru_log_message` OUTPUT_DESCRIPTION

`bru_log` invisibly returns the added log message.

`bru_log_active` returns the previous activation state

Author(s)

Fabian E. Bachl <bachlfab@gmail.com> and Finn Lindgren <finn.lindgren@gmail.com>

Examples

```
## Not run:
if (interactive()) {
  # EXAMPLE1
}
## End(Not run)
```

```

## Not run:
if (interactive()) {
  # EXAMPLE1
}

## End(Not run)
code_runner <- function() {
  oa <- bru_log_active(TRUE)
  on.exit(bru_log_active(oa))
  bru_log("Test message")
}
bru_log_active()
code_runner()
cat(bru_log_get())
bru_log_active()

```

bru_make_stack

Build an inla data stack from linearisation information

Description

Combine linearisation for multiple likelihoods

Usage

```
bru_make_stack(...)
```

```
## S3 method for class 'bru_like'
bru_make_stack(lhood, lin, idx, ...)
```

```
## S3 method for class 'bru_like_list'
bru_make_stack(lhoods, lin, idx, ...)
```

Arguments

...	Arguments passed on to other methods
lhood	A bru_like object
lin	Linearisation information <ul style="list-style-type: none"> • For .bru_like, a linearisation information list with elements A and offset • For .bru_like_list, a list of linearisation information lists
idx	Output from evaluate_index(...)
lhoods	A bru_like_list object

bru_mapper	<i>Methods for bru_mapper objects</i>
------------	---------------------------------------

Description

Methods for bru_mapper objects

Usage

```
bru_mapper(...)  
  
ibm_n(mapper, inla_f = FALSE, ...)  
  
ibm_values(mapper, inla_f = FALSE, ...)  
  
ibm_amatrix(mapper, input, inla_f = FALSE, ...)  
  
ibm_inla_subset(mapper, ...)  
  
ibm_valid_input(mapper, input, inla_f = FALSE, ...)  
  
## Default S3 method:  
bru_mapper(mapper, new_class = NULL, methods = NULL, ...)  
  
## Default S3 method:  
ibm_inla_subset(mapper, ...)  
  
## S3 method for class 'inla.mesh'  
bru_mapper(mesh, ...)  
  
## S3 method for class 'inla.mesh.1d'  
bru_mapper(mesh, indexed = NULL, ...)  
  
bru_mapper_index(n = 1L, ...)  
  
bru_mapper_linear(...)  
  
bru_mapper_matrix(labels, ...)  
  
bru_mapper_factor(values, factor_mapping, ...)  
  
bru_mapper_offset(...)  
  
bru_mapper_multi(mappers, ...)  
  
bru_mapper_collect(mappers, hidden = FALSE, ...)
```

Arguments

...	Arguments passed on to other methods
mapper	A mapper S3 object, normally inheriting from <code>bru_mapper</code> . For the default <code>bru_mapper</code> method, a list that will be converted to a <code>bru_mapper</code> object by adding class information and (optional) methods.
inla_f	logical; when TRUE in <code>ibm_n</code> , <code>ibm_values</code> , and <code>ibm_amatrix</code> methods, these must result in values compatible with <code>INLA::f(...)</code> an specification and corresponding <code>INLA::inla.stack(...)</code> constructions. Implementations do not normally need to do anything different, except for mappers of the type needed for hidden multicomponent models such as "bym2", which can be handled by <code>bru_mapper_collect</code> .
input	The values for which to produce validity information
new_class	If non-NULL, this is added at the front of the class definition
methods,	optional list of named method definitions; See Details.
mesh	An <code>inla.mesh.1d</code> or <code>inla.mesh.2d</code> object to use as a mapper
indexed	logical; If TRUE, the <code>ibm_values()</code> output will be the integer indexing sequence for the latent variables (needed for spde models). If FALSE, the knot locations are returned (useful as an interpolator for rw2 models and similar). Default: NULL, to force user specification of this parameter
n	Size of a model for <code>bru_mapper_index</code>
labels	Column labels for matrix mappings
values	Input values calculated by <code>input_eval.bru_input()</code>
factor_mapping	character; selects the type of factor mapping. <ul style="list-style-type: none"> • 'contrast' for leaving out the first factor level. • 'full' for keeping all levels.
mappers	A list of <code>bru_mapper</code> objects
hidden	logical, set to TRUE to flag that the mapper is to be used as a first level input mapper for <code>INLA::f()</code> in a model that requires making only the first mapper visible to <code>INLA::f()</code> and <code>INLA::inla.stack()</code> , such as for "bym2" models, as activated by the <code>inla_f</code> argument to <code>ibm_n</code> , <code>ibm_values</code> , and <code>ibm_amatrix</code> . Set to FALSE to always access the full mapper, e.g. for rgeneric models

Details

- `bru_mapper` Generic mapper S3 constructor. See below for details of the default constructor that can be used to define new mappers in user code.
- `ibm_n` Generic. Implementations must return the size of the latent vector being mapped to.
- `ibm_values` Generic. Implementations must return a vector that would be interpretable by an `INLA::f(..., values = ...)` specification. The exception is the method for `bru_mapper_multi`, that returns a multi-column data frame

- `ibm_amatrix` Generic. Implementations must return a (sparse) matrix of size `NROW(input)` (except for the `bru_mapper_multi` and `bru_mapper_collect` methods, that require `list()` inputs, and the input size is determined by the combined inputs) by `ibm_n(mapper, inla_f = FALSE)`. The `inla_f=TRUE` argument should only affect the allowed type of input format.
- `ibm_inla_subset` Generic. Implementations must return a logical vector of TRUE/FALSE for the subset such that, given the full A matrix and values output, `A[, subset, drop = FALSE]` and `values[subset]` (or `values[subset, , drop = FALSE]` for data.frame values) are equal to the `inla_f = TRUE` version of A and values. The default method uses the `ibm_values` output to construct the subset indexing.
- `ibm_valid_input` Generic. Implementations must return a logical vector of length `NROW(input)` (or for `bru_mapper_multi` and `bru_mapper_collect` a list of such vectors)
- `bru_mapper.default` adds the "bru_mapper" class and `new_class` to an object. If provided, mapper method functions are added to an environment `.envir` in the object. The generic methods `ibm_n`, `ibm_n_inla`, `ibm_values`, `ibm_values_inla`, `ibm_amatrix`, `ibm_amatrix_inla`, `ibm_valid_input`, and `ibm_valid_input_inla` look for these functions first, and otherwise call `UseMethod()`. This is an alternative to using `.S3method()` to register the methods, e.g. `.S3method("ibm_amatrix", "my_mapper_class", ibm_amatrix.my_mapper_class)`.
- The default `ibm_inla_subset` method uses the `ibm_values` output to construct the inla subset indexing, passing extra arguments such as `multi` on to the methods (this means it supports both regular vector values and `multi=1` data.frame values).
- `bru_mapper_multi` constructs a kronecker product mapping
- `bru_mapper_collect` constructs concatenated collection mapping

See Also

[bru_mapper_methods](#) for specific method implementations.

Examples

```
mapper <- bru_mapper_index(5)
ibm_amatrix(mapper, c(1, 3, 4, 5, 2))
```

bru_mapper_methods *Implementation methods for mapper objects*

Description

A `bru_mapper` sub-class implementation must provide an `ibm_matrix()` method. If the model size 'n' and definition values 'values' are stored in the object itself, default methods are available (see Details). Otherwise the `ibm_n()` and `ibm_values()` methods also need to be provided.

Usage

```
## Default S3 method:
ibm_n(mapper, inla_f = FALSE, ...)

## Default S3 method:
ibm_values(mapper, inla_f = FALSE, ...)

## Default S3 method:
ibm_amatrix(mapper, input, inla_f = FALSE, ...)

## Default S3 method:
ibm_valid_input(mapper, input, ...)

## S3 method for class 'bru_mapper_inla_mesh_2d'
ibm_n(mapper, ...)

## S3 method for class 'bru_mapper_inla_mesh_2d'
ibm_values(mapper, ...)

## S3 method for class 'bru_mapper_inla_mesh_2d'
ibm_amatrix(mapper, input, ...)

## S3 method for class 'bru_mapper_inla_mesh_1d'
ibm_n(mapper, ...)

## S3 method for class 'bru_mapper_inla_mesh_1d'
ibm_values(mapper, ...)

## S3 method for class 'bru_mapper_inla_mesh_1d'
ibm_amatrix(mapper, input, ...)

## S3 method for class 'bru_mapper_index'
ibm_valid_input(mapper, input, ...)

## S3 method for class 'bru_mapper_index'
ibm_amatrix(mapper, input, ...)

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

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

## S3 method for class 'bru_mapper_linear'
ibm_amatrix(mapper, input, ...)

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

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

## S3 method for class 'bru_mapper_matrix'
ibm_amatrix(mapper, input, ...)

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

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

## S3 method for class 'bru_mapper_factor'
ibm_amatrix(mapper, input, ...)

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

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

## S3 method for class 'bru_mapper_offset'
ibm_amatrix(mapper, input, ...)

## S3 method for class 'bru_mapper_multi'
ibm_n(mapper, inla_f = FALSE, multi = 0L, ...)

## S3 method for class 'bru_mapper_multi'
ibm_values(mapper, inla_f = FALSE, multi = 0L, ...)

## S3 method for class 'bru_mapper_multi'
ibm_amatrix(mapper, input, inla_f = FALSE, multi = 0L, ...)

## S3 method for class 'bru_mapper_multi'
ibm_valid_input(mapper, input, inla_f = FALSE, multi = 0L, ...)

## S3 method for class 'bru_mapper_multi'
x[i, drop = TRUE]

## S3 method for class 'bru_mapper_multi'
names(x)

## S3 replacement method for class 'bru_mapper_multi'
names(x) <- value

## S3 method for class 'bru_mapper_collect'
ibm_n(mapper, inla_f = FALSE, multi = 0L, ...)
```

```
## S3 method for class 'bru_mapper_collect'
ibm_values(mapper, inla_f = FALSE, multi = 0L, ...)

## S3 method for class 'bru_mapper_collect'
ibm_amatrix(mapper, input, inla_f = FALSE, multi = 0L, ...)

## S3 method for class 'bru_mapper_collect'
ibm_valid_input(mapper, input, inla_f = FALSE, multi = 0L, ...)

## S3 method for class 'bru_mapper_collect'
x[i, drop = TRUE]

## S3 method for class 'bru_mapper_collect'
names(x)

## S3 replacement method for class 'bru_mapper_collect'
names(x) <- value
```

Arguments

mapper	A mapper S3 object, normally inheriting from <code>bru_mapper</code>
inla_f	logical; when TRUE in <code>ibm_n</code> and <code>ibm_values</code> , these must result in values compatible with <code>INLA::f(...)</code> an specification and corresponding <code>INLA::inla.stack(...)</code> constructions. For <code>ibm_amatrix</code> methods, it may influence how the input data is interpreted. Implementations do not normally need to do anything different, except for mappers of the type needed for hidden multicomponent models such as "bym2", which can be handled by <code>bru_mapper_collect</code> .
...	Arguments passed on to other methods
input	The values for which to produce a mapping matrix
multi	integer or logical; If positive, the number of levels to recurse in a <code>bru_collect_mapper</code> . If TRUE, equivalent to 1L. If FALSE, equivalent to 0L.
x	object from which to extract element(s)
i	indices specifying element(s) to extract
drop	logical; For <code>[.bru_mapper_collect</code> , whether to extract an individual mapper when <code>i</code> identifies a single element. If FALSE, a list of sub-mappers is returned (suitable e.g. for creating a new <code>bru_mapper_collect</code> object). Default: TRUE
value	a character vector of up to the same length as <code>x</code>

Details

- The default `ibm_n()` method returns a non-null element 'n' from the mapper object, and gives an error if it doesn't exist. If `inla_f=TRUE`, first checks for a 'n_inla' element.
- The default `ibm_values()` method returns a non-null element 'values' from the mapper object, and `seq_len(ibm_n(mapper))` if it doesn't exist.

- The default `ibm_amatrix()` gives an error message. Mapper classes must implement their own `ibm_amatrix` method.
- The default `ibm_valid_input()` method returns an all-TRUE logical vector.
- `ibm_amatrix` for `bru_mapper_multi` accepts a list with named entries, or a list with unnamed but ordered elements. The names must match the sub-mappers, see [names.bru_mapper_multi\(\)](#). Each list element should take a format accepted by the corresponding sub-mapper. In case each element is a vector, the input can be given as a `data.frame` with named columns, a matrix with named columns, or a matrix with unnamed but ordered columns.
- `ibm_valid_input` for `bru_mapper_multi` accepts a list with named entries, or a list with unnamed but ordered elements. The names must match the sub-mappers, see [names.bru_mapper_multi\(\)](#). Each list element should take a format accepted by the corresponding sub-mapper. In case each element is a vector, the input can be given as a `data.frame` with named columns, a matrix with named columns, or a matrix with unnamed but ordered columns.
- `ibm_amatrix` for `bru_mapper_collect` accepts a list with named entries, or a list with unnamed but ordered elements. The names must match the sub-mappers, see [names.bru_mapper_collect\(\)](#). Each list element should take a format accepted by the corresponding sub-mapper. In case each element is a vector, the input can be given as a `data.frame` with named columns, a matrix with named columns, or a matrix with unnamed but ordered columns. When `inla_f=TRUE` and `hidden=TRUE` in the mapper definition, the input format should instead match that of the first, non-hidden, sub-mapper.
- `ibm_valid_input` for `bru_mapper_collect` accepts a list with named entries, or a list with unnamed but ordered elements. The names must match the sub-mappers, see [names.bru_mapper_collect\(\)](#). Each list element should take a format accepted by the corresponding sub-mapper. In case each element is a vector, the input can be given as a `data.frame` with named columns, a matrix with named columns, or a matrix with unnamed but ordered columns.

Value

- `[]`-indexing a `bru_mapper_multi` extracts a subset `bru_mapper_multi` object (for `drop FALSE`) or an individual sub-mapper (for `drop TRUE`, and `i` identifies a single element)
- The `names()` method for `bru_mapper_multi` returns the names from the sub-mappers list
- `[]`-indexing a `bru_mapper_collect` extracts a subset `bru_mapper_collect` object (for `drop FALSE`) or an individual sub-mapper (for `drop TRUE`, and `i` identifies a single element)
- The `names()` method for `bru_mapper_collect` returns the names from the sub-mappers list

See Also

[bru_mapper\(\)](#)

bru_options	<i>Create or update an options objects</i>
-------------	--

Description

Create a new options object, or merge information from several objects.

The `_get`, `_set`, and `_reset` functions operate on a global package options override object. In many cases, setting options in specific calls to `bru()` is recommended instead.

Usage

```
bru_options(...)

as.bru_options(x = NULL)

bru_options_default()

bru_options_check(options, ignore_null = TRUE)

bru_options_get(name = NULL, include_default = TRUE)

bru_options_set(..., .reset = FALSE)

bru_options_reset()
```

Arguments

<code>...</code>	A collection of named options, optionally including one or more <code>bru_options</code> objects. Options specified later override the previous options.
<code>x</code>	An object to be converted to an <code>bru_options</code> object.
<code>options</code>	An <code>bru_options</code> object to be checked
<code>ignore_null</code>	Ignore missing or NULL options.
<code>name</code>	Either NULL, or single option name string, or character vector or list with option names, Default: NULL
<code>include_default</code>	logical; If TRUE, the default options are included together with the global override options. Default: TRUE
<code>.reset</code>	For <code>bru_options_set</code> , logical indicating if the global override options list should be emptied before setting the new option(s).

Details

`bru_options_check` checks for valid contents of an `bru_options` object

`bru_options_check()` produces warnings for invalid options.

`bru_options_set()` is used to set global package options.

`bru_options_reset()` clears the global option overrides.

Value

bru_options() returns an bru_options object.

For as.bru_options(), NULL or no input returns an empty bru_options object, a list is converted via bru_options(...), and bru_options input is passed through. Other types of input generates an error.

bru_options_default() returns an bru_options object containing default options.

bru_options_check() returns a logical; TRUE if the object contains valid options for use by other functions

bru_options_get returns either an **bru_options** object, for name == NULL, the contents of single option, if name is a options name string, or a named list of option contents, if name is a list of option name strings.

bru_options_set() returns a copy of the global override options, invisibly (as bru_options_get(include_default = FALSE)).

Valid options

For bru_options and bru_options_set, recognised options are:

bru_verbose logical or numeric; if TRUE, log messages of verbosity ≤ 1 are printed by **bru_log_message()**. If numeric, log messages of verbosity \leq bru_verbose are printed. For line search details, set bru_verbose=2 or 3. Default: 0, to not print any messages

bru_verbose_store logical or numeric; if TRUE, log messages of verbosity ≤ 1 are stored by **bru_log_message()**. If numeric, log messages of verbosity \leq are stored. Default: Inf, to store all messages.

bru_run If TRUE, run inference. Otherwise only return configuration needed to run inference.

bru_max_iter maximum number of inla iterations, default 10. Also see the bru_method\$rel_tol and related options below.

bru_initial An inla object returned from previous calls of INLA::inla, **bru()** or **lgcp()**, or a list of named vectors of starting values for the latent variables. This will be used as a starting point for further improvement of the approximate posterior.

bru_int_args List of arguments passed all the way to the integration method ipoints and int.polygon for 'cp' family models;

method "stable" or "direct". For "stable" (default) integration points are aggregated to mesh vertices.

nsub1 Number of integration points per knot interval in 1D. Default 30.

nsub2 Number of integration points along a triangle edge for 2D. Default 9.

nsub Deprecated parameter that overrides nsub1 and nsub2 if set. Default NULL.

bru_method List of arguments controlling the iterative inlabru method:

taylor 'pandemic' (default, from version 2.1.15).

search Either 'all' (default), to use all available line search methods, or one or more of

'finite' (reduce step size until predictor is finite)

'contract' (decrease step size until trust hypersphere reached)

'expand' (increase step size until no improvement)

'optimise' (fast approximate error norm minimisation)

To disable line search, set to an empty vector. Line search is not available for `taylor="legacy"`.

factor Numeric, > 1 determining the line search step scaling multiplier. Default $(1 + \sqrt{5})/2$.

rel_tol Stop the iterations when the largest change in linearisation point (the conditional latent state mode) in relation to the estimated posterior standard deviation is less than `rel_tol`. Default 0.01 (one percent).

max_step The largest allowed line search step factor. Factor 1 is the full INLA step. Default is 2.

lin_opt_method Which method to use for the line search optimisation step. Default "onestep", using a quadratic approximation based on the value and gradient at zero, and the value at the current best step length guess. The method "full" does line optimisation on the full nonlinear predictor; this is slow and intended for debugging purposes only.

bru_compress_cp logical; when TRUE, compress the $\sum_{i=1}^n \eta_i$ part of the Poisson process likelihood (`family="cp"`) into a single term, with $y = n$, and predictor mean(`eta`). Default: TRUE

inla() options All options not starting with `bru_` are passed on to `inla()`, sometimes after altering according to the needs of the `inlabru` method. Warning: Due to how `inlabru` currently constructs the `inla()` call, the `mean`, `prec`, `mean.intercept`, and `prec.intercept` settings in `control.fixed` will have no effect. Until a more elegant alternative has been implemented, use explicit `mean.linear` and `prec.linear` specifications in each `model="linear"` component instead.

See Also

[bru_options\(\)](#), [bru_options_default\(\)](#), [bru_options_get\(\)](#)

Examples

```
## Not run:
if (interactive()) {
  # Combine global and user options:
  options1 <- bru_options(bru_options_get(), bru_verbose = TRUE)
  # Create a proto-options object in two equivalent ways:
  options2 <- as.bru_options(bru_verbose = TRUE)
  options2 <- as.bru_options(list(bru_verbose = TRUE))
  # Combine options objects:
  options3 <- bru_options(options1, options2)
}

## End(Not run)
## Not run:
if (interactive()) {
  # EXAMPLE1
}

## End(Not run)
## Not run:
if (interactive()) {
  bru_options_check(bru_options(bru_max_iter = "text"))
}
```



```

}

## End(Not run)
## Not run:
if (interactive()) {
  # EXAMPLE1
}

## End(Not run)
## Not run:
if (interactive()) {
  bru_options_set(
    bru_verbose = TRUE,
    verbose = TRUE
  )
}

## End(Not run)

```

bru_safe_inla

Load INLA safely for examples and tests

Description

Loads the INLA package with `requireNamespace("INLA", quietly = TRUE)`, and optionally checks and sets the multicore `num.threads` INLA option.

Usage

```
bru_safe_inla(multicore = NULL, quietly = FALSE)
```

Arguments

<code>multicore</code>	logical; if TRUE, multiple cores are allowed, and the INLA <code>num.threads</code> option is not checked or altered. If FALSE, forces <code>num.threads="1:1"</code> . Default: NULL, checks if running in <code>testthat</code> or non-interactively, in which case sets <code>multicore=FALSE</code> , otherwise TRUE.
<code>quietly</code>	logical; if TRUE, prints diagnostic messages. Default: FALSE.

Value

logical; TRUE if INLA was loaded safely, otherwise FALSE

Examples

```

## Not run:
if (bru_safe_inla()) {
  # Run inla dependent calculations
}

```

```
## End(Not run)
```

```
bru_standardise_names Standardise inla hyperparameter names
```

Description

The inla hyperparameter output uses parameter names that can include whitespace and special characters. This function replaces those characters with underscores.

Usage

```
bru_standardise_names(x)
```

Arguments

x character vector; names to be standardised

Value

A character vector with standardised names

Examples

```
bru_standardise_names("Precision for the Gaussian observations")
```

```
component                    Latent model component construction
```

Description

Similar to `glm()`, `gam()` and `inla()`, `bru()` models can be constructed via a formula-like syntax, where each latent effect is specified. However, in addition to the parts of the syntax compatible with `INLA::inla`, `bru` components offer additional functionality which facilitates modelling, and the predictor expression can be specified separately, allowing more complex and non-linear predictors to be defined. The formula syntax is just a way to allow all model components to be defined in a single line of code, but the definitions can optionally be split up into separate component definitions. See Details for more information.

The component methods all rely on the `component.character()` method, that defines a model component with a given label/name. The user usually doesn't need to call these methods directly, but can instead supply a formula expression that can be interpreted by the `component_list.formula()` method, called inside `bru()`.

Usage

```

component(...)

## S3 method for class 'character'
component(
  object,
  main = NULL,
  weights = NULL,
  ...,
  model = NULL,
  mapper = NULL,
  main_layer = NULL,
  main_selector = NULL,
  n = NULL,
  values = NULL,
  season.length = NULL,
  copy = NULL,
  weights_layer = NULL,
  weights_selector = NULL,
  group = NULL,
  group_mapper = NULL,
  group_layer = NULL,
  group_selector = NULL,
  ngroup = NULL,
  control.group = NULL,
  replicate = NULL,
  replicate_mapper = NULL,
  replicate_layer = NULL,
  replicate_selector = NULL,
  nrep = NULL,
  A.msk = NULL,
  .envir = parent.frame(),
  envir_extra = NULL
)

```

Arguments

...	Parameters passed on to other methods
object	A character label for the component
main	main takes an R expression that evaluates to where the latent variables should be evaluated (coordinates, indices, continuous scalar (for rw2 etc)). Arguments starting with weights, group, replicate behave similarly to main, but for the corresponding features of INLA::f().
weights, weights_layer, weights_selector	Optional specification of effect scaling weights. Same syntax as for main.
model	Either one of "offset", "factor_full", "factor_contrast", "linear", "fixed", or a model name or object accepted by INLA's f function. If set to NULL, then "lin-

	ear" is used for vector inputs, and "fixed" for matrix input (converted internally to an iid model with fixed precision)
mapper	Information about how to do the mapping from the values evaluated in main, and to the latent variables. Auto-detects spde model objects in model and extracts the mesh object to use as the mapper, and auto-generates mappers for indexed models. (Default: NULL, for auto-determination)
main_layer, main_selector	<p>The <code>_layer</code> is a numeric index or character name of which layer/variable to extract from a covariate data object given in main (Default: The effect component name, if it exists in the covariate object, otherwise the first column of the covariate data frame)</p> <p>The <code>_selector</code> is character name of a variable whose contents determines which layer to extract from a covariate for each data point. Overrides the layer. (Default: NULL)</p>
n	The number of latent variables in the model. Should be auto-detected for most or all models (Default: NULL, for auto-detection). An error is given if it can't figure it out by itself.
values	Specifies for what covariate/index values INLA should build the latent model. Normally generated internally based on the mapping details. (Default: NULL, for auto-determination)
season.length	Passed on to INLA: : f() for model "seasonal" (TODO: check if this parameter is still fully handled)
copy	character; label of other component that this component should be a copy of. If the <code>fixed = FALSE</code> , a scaling constant is estimated, via a hyperparameter. If <code>fixed = TRUE</code> , the component scaling is fixed, by default to 1; for fixed scaling, it's more efficient to express the scaling in the predictor expression instead of making a copy component.
group, group_mapper, group_layer, group_selector, ngroup	Optional specification of kronecker/group model indexing.
control.group	list of kronecker/group model parameters, currently passed directly on to INLA: : f
replicate, replicate_mapper, replicate_layer, replicate_selector, nrep	Optional specification of indices for an independent replication model. Same syntax as for main
A.msk	TODO: check/fix/deprecate this parameter. Likely doesn't work at the moment, and I've found no examples that use it.
.envir	Evaluation environment
envir_extra	TODO: check/fix this parameter.

Details

As shorthand, `bru()` will understand basic additive formulae describing fixed effect models. For instance, the components specification $y \sim x$ will define the linear combination of an effect named x and an intercept to the response y with respect to the likelihood family stated when calling `bru()`. Mathematically, the linear predictor η would be written down as

$$\eta = \beta * x + c,$$

where:

- c is the *intercept*
- x is a *covariate*
- β is a *random variable* associated with x and
- $\psi = \beta * x$ is called the *random effect* of x

A problem that arises when using this kind of R formula is that it does not clearly reflect the mathematical formula. For instance, when providing the formula to `inla`, the resulting object will refer to the random effect $\psi = \beta * x$ as x . Hence, it is not clear if x refers to the covariate or the effect of the covariate.

The component `.character` method is `inlabru`'s equivalent to INLA's `f` function but adds functionality that is unique to `inlabru`.

Deprecated parameters:

- `map`: Use `main` instead.
- `mesh`: Use `mapper` instead.

Naming random effects

In INLA, the `f()` notation is used to define more complex models, but a simple linear effect model can also be expressed as

- `formula = y ~ f(x,model = "linear")`,

where `f()` is the `inla` specific function to set up random effects of all kinds. The underlying predictor would again be $\eta = \beta * x + c$ but the result of fitting the model would state x as the random effect's name. `bru` allows rewriting this formula in order to explicitly state the name of the random effect and the name of the associated covariate. This is achieved by replacing `f` with an arbitrary name that we wish to assign to the effect, e.g.

- `components = y ~ psi(x,model = "linear")`.

Being able to discriminate between x and ψ is relevant because of two functionalities `bru` offers. The formula parameters of both `bru()` and the prediction method `predict.bru` are interpreted in the mathematical sense. For instance, `predict` may be used to analyze the analytical combination of the covariate x and the intercept using

- `predict(fit, data.frame(x=2)), ~ exp(psi + Intercept)`.

which corresponds to the mathematical expression $e^{x\beta+c}$.

On the other hand, `predict` may be used to only look at a transformation of the latent variable β_ψ

- `predict(fit,NULL, ~ exp(psi_latent))`.

which corresponds to the mathematical expression e^β .

Author(s)

Fabian E. Bachl <bachlfab@gmail.com> and Finn Lindgren <Finn.Lindgren@gmail.com>

See Also

Other component constructors: [component_list\(\)](#)

Examples

```
# As an example, let us create a linear component. Here, the component is
# called "myLinearEffectOfX" while the covariate the component acts on is
# called "x". Note that a list of components is returned because the
# formula may define multiple components

cmp <- component_list(~ myLinearEffectOfX(main = x, model = "linear"))
summary(cmp)
# Equivalent shortcuts:
cmp <- component_list(~ myLinearEffectOfX(x, model = "linear"))
cmp <- component_list(~ myLinearEffectOfX(x))
# Individual component
cmp <- component("myLinearEffectOfX", main = x, model = "linear")
summary(cmp)

if (bru_safe_inla(quietly = TRUE)) {

  # As an example, let us create a linear component. Here, the component is
  # called "myEffectOfX" while the covariate the component acts on is called "x":

  cmp <- component("myEffectOfX", main = x, model = "linear")
  summary(cmp)

  # A more complicated component:
  cmp <- component("myEffectOfX",
    main = x,
    model = INLA::inla.spde2.matern(INLA::inla.mesh.1d(1:10))
  )

  # Compound fixed effect component, where x and z are in the input data.
  # The formula will be passed on to MatrixModels::model.Matrix:
  cmp <- component("eff", ~ -1 + x:z, model = "fixed")
  summary(cmp)
}
```

component_eval

Evaluate component values in predictor expressions

Description

In predictor expressions, `name_eval(...)` can be used to evaluate the effect of a component called "name".

Usage

```
component_eval(main, group = NULL, replicate = NULL, .state = NULL)
```

Arguments

`main`, `group`, `replicate` Specification of where to evaluate a component. The three inputs are passed on to the respective `bru_mapper` methods.

`.state` The internal component state. Normally supplied automatically by the internal methods for evaluating `inlabru` predictor expressions.

Value

A vector of values for a component

Examples

```
## Not run:
if (bru_safe_inla()) {
  mesh <- INLA::inla.mesh.2d(
    cbind(0, 0),
    offset = 2, max.edge = 0.25
  )
  spde <- INLA::inla.spde2.pcmatern(mesh,
    prior.range = c(0.1, 0.01),
    prior.sigma = c(2, 0.01)
  )
  data <- sp::SpatialPointsDataFrame(
    matrix(runif(10), 5, 2),
    data = data.frame(y = rnorm(5))
  )
  fit <- bru(y ~ -1 + field(coordinates, model = spde),
    family = "gaussian", data = data
  )
  pred <- predict(
    fit,
    data = data.frame(x = 0.5, y = 0.5),
    formula = ~ field_eval(cbind(x, y))
  )
}

## End(Not run)
```

Description

Constructor methods for `inlabru` component lists. Syntax details are given in `component()`.

Usage

```
component_list(object, lhoods = NULL, .envir = parent.frame(), ...)  
  
## S3 method for class 'formula'  
component_list(object, lhoods = NULL, .envir = parent.frame(), ...)  
  
## S3 method for class 'list'  
component_list(object, lhoods = NULL, .envir = parent.frame(), ...)  
  
## S3 method for class 'component_list'  
c(...)  
  
## S3 method for class 'component_list'  
x[i]
```

Arguments

object	The object to operate on
lhoods	A <code>bru_like_list</code> object
.envir	An evaluation environment for non-formula input
...	Parameters passed on to other methods. Also see <code>Details</code> .
x	<code>component_list</code> object from which to extract element(s)
i	indices specifying elements to extract

Details

- `component_list.formula`: Convert a component formula into a `component_list` object
- `component_list.list`: Combine a list of components and/or component formulas into a `component_list` object
- `c.component_list`: The ... arguments should be `component_list` objects. The environment from the first argument will be applied to the resulting list.

Author(s)

Fabian E. Bachl <bachlfab@gmail.com> and Finn Lindgren <finn.lindgren@gmail.com>

See Also

Other component constructors: [component\(\)](#)
Other component constructors: [component\(\)](#)

Examples

```
# As an example, let us create a linear component. Here, the component is
# called "myLinearEffectOfX" while the covariate the component acts on is
# called "x". Note that a list of components is returned because the
# formula may define multiple components

eff <- component_list(~ myLinearEffectOfX(main = x, model = "linear"))
summary(eff[[1]])
# Equivalent shortcuts:
eff <- component_list(~ myLinearEffectOfX(x, model = "linear"))
eff <- component_list(~ myLinearEffectOfX(x))
# Individual component
eff <- component("myLinearEffectOfX", main = x, model = "linear")
```

cprod

Cross product of integration points

Description

Calculates the cross product of integration points in different dimensions and multiplies their weights accordingly. If the object defining points in a particular dimension has no weights attached to it all weights are assumed to be 1.

Usage

```
cprod(...)
```

Arguments

```
...          data.frame or SpatialPointsDataFrame objects, each one usually obtained
              by a call to the ipoints function.
```

Value

A data.frame or SpatialPointsDataFrame of multidimensional integration points and their weights

Author(s)

Fabian E. Bachl <bachlfab@gmail.com>

Examples

```
# ipoints needs INLA
if (bru_safe_inla()) {
  # Create integration points in dimension 'myDim' and 'myDiscreteDim'
  ips1 <- ipoints(rbind(c(0, 3), c(3, 8)), 17, name = "myDim")
  ips2 <- ipoints(domain = c(1, 2, 4), name = "myDiscreteDim")
}
```

```

# Calculate the cross product
ips <- cprod(ips1, ips2)

# Plot the integration points
plot(ips$myDim, ips$myDiscreteDim, cex = 10 * ips$weight)
}

```

deltaIC

Summarise DIC and WAIC from lgcp objects.

Description

Calculates DIC and/or WAIC differences and produces an ordered summary.

Usage

```
deltaIC(..., criterion = "DIC")
```

Arguments

... Comma-separated objects inheriting from class `inla` and obtained from a run of `INLA::inla()`, `bru()` or `lgcp()`

criterion character vector. If it includes 'DIC', computes DIC differences; If it contains 'WAIC', computes WAIC differences. Default: 'DIC'

Value

A data frame with each row containing the Model name, DIC and Delta.DIC, and/or WAIC and Delta.WAIC.

Examples

```

if (bru_safe_inla(multicore = FALSE)) {
  # Generate some data
  input.df <- data.frame(idx = 1:10, x = cos(1:10))
  input.df <- within(
    input.df,
    y <- rpois(10, 5 + 2 * cos(1:10)) + rnorm(10, mean = 0, sd = 0.1))
  )

  # Fit two models
  fit1 <- bru(y ~ x, family = "poisson", data = input.df)
  fit2 <- bru(y ~ x + rand(idx, model = "iid"), family = "poisson", data = input.df)

  # Compare DIC

```

```

    deltaIC(fit1, fit2)
  }

```

devel.cvmeasure	<i>Variance and correlations measures for prediction components</i>
-----------------	---

Description

Calculates local and integrated variance and correlation measures as introduced by Yuan et al. (2017).

Usage

```
devel.cvmeasure(joint, prediction1, prediction2, samplers = NULL, mesh = NULL)
```

Arguments

joint	A joint prediction of two latent model components.
prediction1	A prediction of first component.
prediction2	A prediction of the first component.
samplers	A SpatialPolygon object describing the area for which to compute the cumulative variance measure.
mesh	The inla.mesh for which the prediction was performed (required for cumulative Vmeasure).

Value

Variance and correlations measures.

References

Y. Yuan, F. E. Bachl, F. Lindgren, D. L. Brochers, J. B. Illian, S. T. Buckland, H. Rue, T. Gerrodette. 2017. Point process models for spatio-temporal distance sampling data from a large-scale survey of blue whales. <https://arxiv.org/abs/1604.06013>

Examples

```

if (bru_safe_inla() && require(ggplot2, quietly = TRUE)) {
  # Load Gorilla data
  data("gorillas", package = "inlabru")
  # Use RColorBrewer
  library(RColorBrewer)

```

```

# Fit a model with two components:
# 1) A spatial smooth SPDE
# 2) A spatial covariate effect (vegetation)

pcmatern <- INLA::inla.spde2.pcmatern(gorillas$mesh,
  prior.sigma = c(0.1, 0.01),
  prior.range = c(0.01, 0.01)
)

cmp <- coordinates ~ vegetation(gorillas$gcov$vegetation, model = "factor_contrast") +
  spde(coordinates, model = pcmatern) -
  Intercept(1)

fit <- lgcp(cmp, gorillas$neats,
  samplers = gorillas$boundary,
  domain = list(coordinates = gorillas$mesh),
  options = list(control.inla = list(int.strategy = "eb"))
)

# Predict SPDE and vegetation at the mesh vertex locations

vrt <- vertices(gorillas$mesh)
pred <- predict(
  fit,
  vrt,
  ~ list(
    joint = spde + vegetation,
    field = spde,
    veg = vegetation
  )
)

# Plot component mean

multiplot(ggplot() +
  gg(gorillas$mesh, color = pred$joint$mean) +
  coord_equal() +
  theme(legend.position = "bottom"),
  ggplot() +
  gg(gorillas$mesh, color = pred$field$mean) +
  coord_equal() +
  theme(legend.position = "bottom"),
  ggplot() +
  gg(gorillas$mesh, color = pred$veg$mean) +
  coord_equal() +
  theme(legend.position = "bottom"),
  cols = 3
)

# Plot component variance

multiplot(ggplot() +

```

```

    gg(gorillas$mesh, color = pred$joint$var) +
    coord_equal() +
    theme(legend.position = "bottom"),
  ggplot() +
    gg(gorillas$mesh, color = pred$field$var) +
    coord_equal() +
    theme(legend.position = "bottom"),
  ggplot() +
    gg(gorillas$mesh, color = pred$veg$var) +
    coord_equal() +
    theme(legend.position = "bottom"),
  cols = 3
)

# Calculate variance and correlation measure

vm <- devel.cvmeasure(pred$joint, pred$field, pred$veg)
lprange <- range(vm$var.joint, vm$var1, vm$var2)

# Variance contribution of the components

csc <- scale_fill_gradientn(colours = brewer.pal(9, "YlOrRd"), limits = lprange)
boundary <- gorillas$boundary

plot.1 <- ggplot() +
  gg(gorillas$mesh, color = vm$var.joint, mask = boundary) +
  csc +
  coord_equal() +
  ggtitle("joint") +
  theme(legend.position = "bottom")
plot.2 <- ggplot() +
  gg(gorillas$mesh, color = vm$var1, mask = boundary) +
  csc +
  coord_equal() +
  ggtitle("SPDE") +
  theme(legend.position = "bottom")
plot.3 <- ggplot() +
  gg(gorillas$mesh, color = vm$var2, mask = boundary) +
  csc +
  coord_equal() +
  ggtitle("vegetation") +
  theme(legend.position = "bottom")

multiplot(plot.1, plot.2, plot.3, cols = 3)

# Covariance of SPDE field and vegetation

ggplot() +
  gg(gorillas$mesh, color = vm$cov)

# Correlation between field and vegetation

ggplot() +

```

```

gg(gorillas$mesh, color = vm$cor)

# Variance and correlation integrated over space

vm.int <- devel.cvmeasure(pred$joint, pred$field, pred$veg,
  samplers = ipoints(gorillas$boundary, gorillas$mesh),
  mesh = gorillas$mesh
)
vm.int
}

```

evaluate_A	<i>Compute all A-matrices</i>
------------	-------------------------------

Description

Computes the A matrices for included components for each model likelihood

Usage

```
evaluate_A(model, lhoods, inla_f)
```

Arguments

model	A bru_model object
lhoods	A bru__like_list object
inla_f	logical

evaluate_index	<i>Compute all index values</i>
----------------	---------------------------------

Description

Computes the index values matrices for included components

Usage

```
evaluate_index(model, lhoods)
```

Arguments

model	A bru_model object
lhoods	A bru__like_list object

Value

A named list of idx_full and idx_inla, named list of indices, and inla_subset, and inla_subset, a named list of logical subset specifications for extracting the INLA::f() compatible index subsets.

expand_labels	<i>Expand labels</i>
---------------	----------------------

Description

Expand labels

Usage

```
expand_labels(labels, expand, suffix)
```

Arguments

labels	character vector; original labels
expand	character vector; subset of labels to expand
suffix	character; the suffix to add to the labels selected by expand

Value

a vector of labels with suffix appended to the selected labels

fm_CRS	<i>Create a coordinate reference system object</i>
--------	--

Description

Creates either a CRS object or an `inla.CRS` object, describing a coordinate reference system

Usage

```
fm_CRS(  
  projargs = NULL,  
  doCheckCRSArgs = TRUE,  
  args = NULL,  
  oblique = NULL,  
  SRS_string = NULL,  
  ...  
)  
  
fm_wkt_predef()
```

Arguments

projargs	Either 1) a projection argument string suitable as input to <code>sp::CRS</code> , or 2) an existing CRS object, or 3) a shortcut reference string to a predefined projection; run <code>names(fm_wkt_predef())</code> for valid predefined projections.
doCheckCRSArgs	default TRUE, must be set to FALSE by package developers including CRS in an S4 class definition to avoid uncontrollable loading of the <code>rgdal</code> namespace.
args	An optional list of name/value pairs to add to and/or override the PROJ4 arguments in <code>projargs</code> . <code>name=value</code> is converted to <code>"name=value"</code> , and <code>name=NA</code> is converted to <code>"name"</code> .
oblique	Vector of length at most 4 of rotation angles (in degrees) for an oblique projection, all values defaulting to zero. The values indicate (longitude, latitude, orientation, orbit), as explained in the Details section below.
SRS_string	a WKT2 string defining the coordinate system; see <code>sp::CRS</code> . This takes precedence over <code>projargs</code> .
...	Additional parameters. Not currently in use.

Details

The first two elements of the oblique vector are the (longitude, latitude) coordinates for the oblique centre point. The third value (orientation) is a counterclockwise rotation angle for an observer looking at the centre point from outside the sphere. The fourth value is the quasi-longitude (orbit angle) for a rotation along the oblique observers equator.

Simple oblique: `oblique=c(0, 45)`

Polar: `oblique=c(0, 90)`

Quasi-transversal: `oblique=c(0, 0, 90)`

Satellite orbit viewpoint: `oblique=c(lon0-time*v1, 0, orbitangle, orbit0+time*v2)`, where `lon0` is the longitude at which a satellite orbit crosses the equator at `time=0`, when the satellite is at an angle `orbit0` further along in its orbit. The orbital angle relative to the equatorial plane is `orbitangle`, and `v1` and `v2` are the angular velocities of the planet and the satellite, respectively. Note that "forward" from the satellite's point of view is "to the right" in the projection.

When `oblique[2]` or `oblique[3]` are non-zero, the resulting projection is only correct for perfect spheres.

Value

Either an `sp::CRS` object or an `inla.CRS` object, depending on if the coordinate reference system described by the parameters can be expressed with a pure `sp::CRS` object or not.

An S3 `inla.CRS` object is a list, usually (but not necessarily) containing at least one element:

`crs` The basic `sp::CRS` object

`fm_wkt_predef` returns a WKT2 string defining a projection

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

[sp::CRS\(\)](#), [fm_crs_wkt](#), [fm_sp_get_crs\(\)](#), [fm_identical_CRS\(\)](#)

Examples

```
if (require(rgdal)) {
  if (fm_has_PROJ6()) {
    crs1 <- fm_CRS("longlat_globe")
    crs2 <- fm_CRS("lambert_globe")
    crs3 <- fm_CRS("mollweide_norm")
    crs4 <- fm_CRS("hammer_globe")
    crs5 <- fm_CRS("sphere")
    crs6 <- fm_CRS("globe")
  } else {
    # Old definitions for pre-PROJ6:
    # Old radius-1 projections have a added "_norm" in the PROJ6 version of
    # the fm_CRS() predefined projections. They are detected and converted
    # to the new versions when PROJ6 is available.
    crs1 <- fm_CRS("longlat") # PROJ6: longlat_norm
    crs2 <- fm_CRS("lambert") # PROJ6: lambert_norm
    crs3 <- fm_CRS("mollweide") # PROJ6: mollweide_norm
    crs4 <- fm_CRS("hammer") # PROJ6: hammer_norm
    crs5 <- fm_CRS("sphere")
    crs6 <- fm_CRS("globe")
  }
}
## Not run:
names(fm_wkt_predef())

## End(Not run)
```

fm_has_PROJ6

PROJ6 detection

Description

Detect whether PROJ6 is available

Usage

```
fm_has_PROJ6()
```

```
fm_not_for_PROJ6(fun = NULL)
```

```
fm_not_for_PROJ4(fun = NULL)
```

```
fm_fallback_PROJ6(fun = NULL)
```

```
fm_requires_PROJ6(fun = NULL)
```

Arguments

fun The name of the function that requires PROJ6. Default: NULL, which uses the name of the calling function.

Details

fm_not_for_PROJ6 is called to warn about using old PROJ4 features even though PROJ6 is available

fm_not_for_PROJ4 is called to give an error when calling methods that are only available for PROJ6

fm_fallback_PROJ6 is called to warn about falling back to using old PROJ4 methods when a PROJ6 method hasn't been implemented

fm_requires_PROJ6 is called to give an error when PROJ6 is required but not available

Value

For fm_has_PROJ6, logical; TRUE if PROJ6 is available, FALSE otherwise

Examples

```
fm_has_PROJ6()
```

fm_spTransform	<i>Handle transformation of various inla objects according to coordinate reference systems of sp::CRS or INLA::inla.CRS class.</i>
----------------	--

Description

Handle transformation of various inla objects according to coordinate reference systems of sp::CRS or INLA::inla.CRS class.

Usage

```
fm_spTransform(x, ...)

## Default S3 method:
fm_spTransform(x, crs0, crs1, passthrough = FALSE, ...)

## S3 method for class 'SpatialPoints'
fm_spTransform(x, CRSobj, passthrough = FALSE, ...)

## S3 method for class 'SpatialPointsDataFrame'
fm_spTransform(x, CRSobj, passthrough = FALSE, ...)

## S3 method for class 'inla.mesh.lattice'
fm_spTransform(x, CRSobj, passthrough = FALSE, ...)
```

```
## S3 method for class 'inla.mesh.segment'
fm_spTransform(x, CRSobj, passthrough = FALSE, ...)

## S3 method for class 'inla.mesh'
fm_spTransform(x, CRSobj, passthrough = FALSE, ...)
```

Arguments

x	The object that should be transformed from it's current CRS to a new CRS
...	Potential additional arguments
crs0	The source sp::CRS or inla.CRS object
crs1	The target sp::CRS or inla.CRS object
passthrough	Default is FALSE. Setting to TRUE allows objects with no CRS information to be passed through without transformation.
CRSobj	The target sp::CRS or inla.CRS object

Details

The default method handles low level transformation of raw coordinates.

See Also

[fm_CRS\(\)](#)

fm_sp_get_crs	<i>Extract CRS information</i>
---------------	--------------------------------

Description

Wrapper for CRS(projargs) (PROJ4) and CRS(wkt) for sp::Spatial objects.

Usage

```
fm_sp_get_crs(x)
```

Arguments

x	A sp::Spatial object
---	----------------------

Details

This function is a convenience method to workaround PROJ4/PROJ6 differences, and the lack of a crs extraction method for Spatial objects.

Value

A CRS object, or NULL if no valid CRS identified

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

Examples

```
## Not run:
if (interactive()) {
  s <- sp::SpatialPoints(matrix(1:6, 3, 2), proj4string = fm_CRS("sphere"))
  fm_sp_get_crs(s)
}

## End(Not run)
```

fm_wkt_as_wkt_tree *Internal WKT handling*

Description

Conversion between WKT and a tree representation

Usage

```
fm_wkt_as_wkt_tree(x, ...)

fm_wkt_tree_as_wkt(x, pretty = FALSE, ...)

fm_wkt_tree_get_item(x, item, duplicate = 1)

fm_wkt_tree_set_item(x, item_tree, duplicate = 1)
```

Arguments

x	A WKT2 string, or a wkt_tree list structure
...	Unused
pretty	logical; If TRUE, use pretty formatting. Default: FALSE
item	character vector with item labels identifying a parameter item entry.
duplicate	For items that have more than one match, duplicate indicates the index number of the desired version. Default: 1
item_tree	An item tree identifying a parameter item entry

 fm_wkt_is_geocent *Handling CRS/WKT*

Description

Get and set CRS object or WKT string properties.

Usage

```

fm_wkt_is_geocent(wkt)
fm_crs_is_geocent(crs)
fm_wkt_get_ellipsoid_radius(wkt)
fm_crs_get_ellipsoid_radius(crs)
fm_wkt_set_ellipsoid_radius(wkt, radius)
fm_crs_set_ellipsoid_radius(crs, radius)
fm_wkt_unit_params()
fm_wkt_get_lengthunit(wkt)
fm_wkt_set_lengthunit(wkt, unit, params = NULL)
fm_crs_get_wkt(crs)
fm_crs_get_lengthunit(crs)
fm_crs_set_lengthunit(crs, unit, params = NULL)

```

Arguments

wkt	A WKT2 character string
crs	A sp::CRS or inla.CRS object
radius	numeric; The new radius value
unit	character, name of a unit. Supported names are "metre", "kilometre", and the aliases "meter", "m", "International metre", "kilometer", and "km", as defined by fm_wkt_unit_params or the params argument. (For legacy PROJ4 use, only "m" and "km" are supported)
params	Length unit definitions, in the list format produced by fm_wkt_unit_params(), Default: NULL, which invokes fm_wkt_unit_params()

Value

For `fm_wkt_unit_params`, a list of named unit definitions

For `fm_wkt_get_lengthunit`, a list of length units used in the wkt string, excluding the ellipsoid radius unit.

For `fm_wkt_set_lengthunit`, a WKT2 string with altered length units. Note that the length unit for the ellipsoid radius is unchanged.

For `fm_crs_get_wkt`, WKT2 string.

For `fm_crs_get_lengthunit`, a list of length units used in the wkt string, excluding the ellipsoid radius unit. (For legacy PROJ4 code, the raw units from the proj4string are returned, if present.)

For `fm_crs_set_lengthunit`, a `sp::CRS` object with altered length units. Note that the length unit for the ellipsoid radius is unchanged.

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

[fm_sp_get_crs\(\)](#)

Examples

```
## Not run:
if (fm_has_PROJ6()) {
  c1 <- fm_CRS("globe")
  fm_crs_get_lengthunit(c1)
  c2 <- fm_crs_set_lengthunit(c1, "km")
  fm_crs_get_lengthunit(c2)
}

## End(Not run)
```

generate

Generate samples from fitted bru models

Description

Generic function for sampling for fitted models. The function invokes particular methods which depend on the class of the first argument.

Takes a fitted bru object produced by the function `bru()` and produces samples given a new set of values for the model covariates or the original values used for the model fit. The samples can be based on any R expression that is valid given these values/covariates and the joint posterior of the estimated random effects.

Usage

```
generate(object, ...)

## S3 method for class 'bru'
generate(
  object,
  data = NULL,
  formula = NULL,
  n.samples = 100,
  seed = 0L,
  num.threads = NULL,
  include = NULL,
  exclude = NULL,
  ...
)
```

Arguments

object	A bru object obtained by calling <code>bru()</code> .
...	additional, unused arguments.
data	A <code>data.frame</code> or <code>SpatialPointsDataFrame</code> of covariates needed for sampling.
formula	A formula where the right hand side defines an R expression to evaluate for each generated sample. If <code>NULL</code> , the latent and hyperparameter states are returned as named list elements. See Details for more information.
n.samples	Integer setting the number of samples to draw in order to calculate the posterior statistics. The default, 100, is rather low but provides a quick approximate result.
seed	Random number generator seed passed on to <code>INLA::inla.posterior.sample</code>
num.threads	Specification of desired number of threads for parallel computations. Default <code>NULL</code> , leaves it up to INLA. When <code>seed != 0</code> , overridden to "1:1"
include	Character vector of component labels that are needed by the predictor expression; Default: <code>NULL</code> (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: <code>NULL</code> (do not remove any components from the inclusion list)

Details

In addition to the component names (that give the effect of each component evaluated for the input data), the suffix `_latent` variable name can be used to directly access the latent state for a component, and the suffix function `_eval` can be used to evaluate a component at other input values than the expressions defined in the component definition itself, e.g. `field_eval(cbind(x,y))` for a component that was defined with `field(coordinates,...)` (see also `component_eval()`).

For "iid" models with `mapper = bru_mapper_index(n)`, `rnorm()` is used to generate new realisations for indices greater than `n`.

Value

The form of the value returned by `gg` depends on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

List of generated samples

See Also

[predict.bru](#)

Examples

```

if (bru_safe_inla(multicore = FALSE)) {

  # Generate data for a simple linear model

  input.df <- data.frame(x = cos(1:10))
  input.df <- within(input.df, y <- 5 + 2 * cos(1:10) + rnorm(10, mean = 0, sd = 0.1))

  # Fit the model

  fit <- bru(y ~ xeff(main = x, model = "linear"),
            family = "gaussian", data = input.df
            )
  summary(fit)

  # Generate samples for some predefined x

  df <- data.frame(x = seq(-4, 4, by = 0.1))
  smp <- generate(fit, df, ~ xeff + Intercept, n.samples = 10)

  # Plot the resulting realizations

  plot(df$x, smp[, 1], type = "l")
  for (k in 2:ncol(smp)) points(df$x, smp[, k], type = "l")

  # We can also draw samples form the joint posterior

  df <- data.frame(x = 1)
  smp <- generate(fit, df, ~ data.frame(xeff, Intercept), n.samples = 10)
  smp[[1]]

  # ... and plot them
  if (require(ggplot2, quietly = TRUE)) {
    plot(do.call(rbind, smp))
  }
}

if (bru_safe_inla(multicore = FALSE)) {

```



```
# Generate data for a simple linear model

input.df <- data.frame(x = cos(1:10))
input.df <- within(input.df, y <- 5 + 2 * cos(1:10) + rnorm(10, mean = 0, sd = 0.1))

# Fit the model

fit <- bru(y ~ xeff(main = x, model = "linear"),
  family = "gaussian", data = input.df
)
summary(fit)

# Generate samples for some predefined x

df <- data.frame(x = seq(-4, 4, by = 0.1))
smp <- generate(fit, df, ~ xeff + Intercept, n.samples = 10)

# Plot the resulting realizations

plot(df$x, smp[, 1], type = "l")
for (k in 2:ncol(smp)) points(df$x, smp[, k], type = "l")

# We can also draw samples from the joint posterior

df <- data.frame(x = 1)
smp <- generate(fit, df, ~ data.frame(xeff, Intercept), n.samples = 10)
smp[[1]]

# ... and plot them
if (require(ggplot2, quietly = TRUE)) {
  plot(do.call(rbind, smp))
}
}
```

Description

`gg` is a generic function for generating geomes from various kinds of spatial objects, e.g. Spatial* data, meshes, Raster objects and inla/inlabru predictions. The function invokes particular methods which depend on the [class](#) of the first argument.

Usage

```
gg(data, ...)
```

Arguments

data an object for which to generate a geom.
 ... Arguments passed on to the geom method.

Value

The form of the value returned by gg depends on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

See Also

Other geomes for inla and inlabru predictions: [gg.data.frame\(\)](#), [gg.matrix\(\)](#), [gg.prediction\(\)](#), [gm\(\)](#)

Other geomes for spatial data: [gg.SpatialGridDataFrame\(\)](#), [gg.SpatialLines\(\)](#), [gg.SpatialPixelsDataFrame\(\)](#), [gg.SpatialPixels\(\)](#), [gg.SpatialPoints\(\)](#), [gg.SpatialPolygons\(\)](#), [gm\(\)](#)

Other geomes for meshes: [gg.inla.mesh.1d\(\)](#), [gg.inla.mesh\(\)](#), [gm\(\)](#)

Other geomes for Raster data: [gg.RasterLayer\(\)](#), [gm\(\)](#)

Examples

```
if (require("ggplot2", quietly = TRUE) &&
    require(ggpolypath, quietly = TRUE)) {
  # Load Gorilla data

  data(gorillas, package = "inlabru")

  # Invoke ggplot and add geomes for the Gorilla nests and the survey boundary

  ggplot() +
    gg(gorillas$boundary) +
    gg(gorillas$nests)
}
```

gg.data.frame

Geom for data.frame

Description

This geom constructor will simply call [gg.prediction](#) for the data provided.

Usage

```
## S3 method for class 'data.frame'
gg(...)
```

Arguments

... Arguments passed on to [gg.prediction\(\)](#).

Details

Requires the ggplot2 package.

Value

Concatenation of a geom_line value and optionally a geom_ribbon value.

See Also

Other geomes for inla and inlabru predictions: [gg.matrix\(\)](#), [gg.prediction\(\)](#), [gg\(\)](#), [gm\(\)](#)

Examples

```
if (bru_safe_inla() && require(ggplot2, quietly = TRUE)) {
  # Generate some data

  input.df <- data.frame(x = cos(1:10))
  input.df <- within(input.df, y <- 5 + 2 * cos(1:10) + rnorm(10, mean = 0, sd = 0.1))

  # Fit a model with fixed effect 'x' and intercept 'Intercept'

  fit <- bru(y ~ x, family = "gaussian", data = input.df)

  # Predict posterior statistics of 'x'

  xpost <- predict(fit, data = NULL, formula = ~x_latent)

  # The statistics include mean, standard deviation, the 2.5% quantile, the median,
  # the 97.5% quantile, minimum and maximum sample drawn from the posterior as well as
  # the coefficient of variation and the variance.

  xpost

  # For a single variable like 'x' the default plotting method invoked by gg() will
  # show these statistics in a fashion similar to a box plot:
  ggplot() +
    gg(xpost)

  # The predict function can also be used to simulataneously estimate posteriors
  # of multiple variables:

  xipost <- predict(fit,
    data = NULL,
    formula = ~ c(
      Intercept = Intercept_latent,
      x = x_latent
    )
  )
  xipost
```

```

# If we still want a plot in the previous style we have to set the bar parameter to TRUE

p1 <- ggplot() +
  gg(xipost, bar = TRUE)
p1

# Note that gg also understands the posterior estimates generated while running INLA

p2 <- ggplot() +
  gg(fit$summary.fixed, bar = TRUE)
multiplot(p1, p2)

# By default, if the prediction has more than one row, gg will plot the column 'mean' against
# the row index. This is for instance usefuul for predicting and plotting function
# but not very meaningful given the above example:

ggplot() +
  gg(xipost)

# For ease of use we can also type

plot(xipost)

# This type of plot will show a ribbon around the mean, which viszualizes the upper and lower
# quantiles mentioned above (2.5 and 97.5%). Plotting the ribbon can be turned of using the
# \code{ribbon} parameter

ggplot() +
  gg(xipost, ribbon = FALSE)

# Much like the other geomes produced by gg we can adjust the plot using ggplot2 style
# commands, for instance

ggplot() +
  gg(xipost) +
  gg(xipost, mapping = aes(y = median), ribbon = FALSE, color = "red")
}

```

Description

This function extracts the graph of an `inla.mesh` object and uses `geom_line` to visualize the graph's edges. Alternatively, if the `color` argument is provided, interpolates the colors across for a set of `SpatialPixels` covering the mesh area and calls `gg.SpatialPixelsDataFrame()` to plot the interpolation. Requires the `ggplot2` package.

Usage

```
## S3 method for class 'inla.mesh'
gg(
  data,
  color = NULL,
  alpha = NULL,
  edge.color = "grey",
  interior = TRUE,
  int.color = "blue",
  exterior = TRUE,
  ext.color = "black",
  crs = NULL,
  mask = NULL,
  nx = 500,
  ny = 500,
  ...
)
```

Arguments

<code>data</code>	An <code>INLA::inla.mesh</code> object.
<code>color</code>	A vector of scalar values to fill the mesh with colors. The length of the vector must correspond to the number of mesh vertices. The alternative name <code>colour</code> is also recognised.
<code>alpha</code>	A vector of scalar values setting the alpha value of the colors provided.
<code>edge.color</code>	Color of the mesh edges.
<code>interior</code>	If <code>TRUE</code> , plot the interior boundaries of the mesh.
<code>int.color</code>	Color used to plot the interior boundaries.
<code>exterior</code>	If <code>TRUE</code> , plot the exterior boundaries of the mesh.
<code>ext.color</code>	Color used to plot the interior boundaries.
<code>crs</code>	A CRS object defining the coordinate system to project the mesh to before plotting.
<code>mask</code>	A <code>SpatialPolygon</code> defining the region that is plotted.
<code>nx</code>	Number of pixels in x direction (when plotting using the <code>color</code> parameter).
<code>ny</code>	Number of pixels in y direction (when plotting using the <code>color</code> parameter).
<code>...</code>	ignored arguments (S3 generic compatibility).

Value

`geom_line` return values or, if the `color` argument is used, the values of `gg.SpatialPixelsDataFrame()`.

See Also

Other geomes for meshes: [gg.inla.mesh.1d\(\)](#), [gg\(\)](#), [gm\(\)](#)

Examples

```

if (bru_safe_inla() &&
    require(ggplot2, quietly = TRUE) &&
    require(ggpolypath, quietly = TRUE)) {

  # Load Gorilla data
  data("gorillas", package = "inlabru")

  # Plot mesh using default edge colors

  ggplot() +
    gg(gorillas$mesh)

  # Don't show interior and exterior boundaries

  ggplot() +
    gg(gorillas$mesh, interior = FALSE, exterior = FALSE)

  # Change the edge colors

  ggplot() +
    gg(gorillas$mesh,
       edge.color = "green",
       int.color = "black",
       ext.color = "blue"
    )

  # Use the x-coordinate of the vertices to colorize the triangles and
  # mask the plotted area by the survey boundary, i.e. only plot the inside

  xcoord <- gorillas$mesh$loc[, 1]
  ggplot() +
    gg(gorillas$mesh, color = (xcoord - 580), mask = gorillas$boundary) +
    gg(gorillas$boundary)
}

```

gg.inla.mesh.1d

Geom for inla.mesh.1d objects

Description

This function generates a `geom_point` object showing the knots (vertices) of a 1D mesh. Requires the `ggplot2` package.

Usage

```
## S3 method for class 'inla.mesh.1d'
```

```
gg(  
  data,  
  mapping = ggplot2::aes(.data[["x"]], .data[["y"]]),  
  y = 0,  
  shape = 4,  
  ...  
)
```

Arguments

data	An inla.mesh.1d object.
mapping	aesthetic mappings created by aes. These are passed on to geom_point.
y	Single or vector numeric defining the y-coordinates of the mesh knots to plot.
shape	Shape of the knot markers.
...	parameters passed on to geom_point.

Value

An object generated by geom_point.

See Also

Other geomes for meshes: [gg.inla.mesh\(\)](#), [gg\(\)](#), [gm\(\)](#)

Examples

```
# Some features use the INLA package.  
if (require("INLA", quietly = TRUE) &&  
    require("ggplot2", quietly = TRUE)) {  
  
  # Create a 1D mesh  
  
  mesh <- inla.mesh.1d(seq(0, 10, by = 0.5))  
  
  # Plot it  
  
  ggplot() +  
    gg(mesh)  
  
  # Plot it using a different shape and size for the mesh nodes  
  
  ggplot() +  
    gg(mesh, shape = "|", size = 5)  
}
```

`gg.matrix`*Geom for matrix*

Description

Creates a tile geom for plotting a matrix

Usage

```
## S3 method for class 'matrix'  
gg(data, mapping = NULL, ...)
```

Arguments

`data` A matrix object.

`mapping` a set of aesthetic mappings created by `aes`. These are passed on to `geom_tile`.

`...` Arguments passed on to `geom_tile`.

Details

Requires the `ggplot2` package.

Value

A `geom_tile` with reversed y scale.

See Also

Other geomes for `inla` and `inlabru` predictions: [gg.data.frame\(\)](#), [gg.prediction\(\)](#), [gg\(\)](#), [gm\(\)](#)

Examples

```
if (require("ggplot2", quietly = TRUE)) {  
  A <- matrix(runif(100), nrow = 10)  
  ggplot() +  
    gg(A)  
}
```

gg.prediction	<i>Geom for predictions</i>
---------------	-----------------------------

Description

This geom serves to visualize prediction objects which usually results from a call to `predict.bru()`. Predictions objects provide summary statistics (mean, median, sd, ...) for one or more random variables. For single variables (or if requested so by setting `bar = TRUE`), a boxplot-style geom is constructed to show the statistics. For multivariate predictions the mean of each variable (y-axis) is plotted against the row number of the variable in the prediction data frame (x-axis) using `geom_line`. In addition, a `geom_ribbon` is used to show the confidence interval.

Note: `gg.prediction` also understands the format of INLA-style posterior summaries, e.g. `fit$summary.fixed` for an `inla` object fit

Requires the `ggplot2` package.

Usage

```
## S3 method for class 'prediction'
gg(data, mapping = NULL, ribbon = TRUE, alpha = 0.3, bar = FALSE, ...)
```

Arguments

<code>data</code>	A prediction object, usually the result of a <code>predict.bru()</code> call.
<code>mapping</code>	a set of aesthetic mappings created by <code>aes</code> . These are passed on to <code>geom_line</code> .
<code>ribbon</code>	If <code>TRUE</code> , plot a ribbon around the line based on the upper and lower 2.5 percent quantiles.
<code>alpha</code>	The ribbons numeric alpha level in <code>[0,1]</code> .
<code>bar</code>	If <code>TRUE</code> plot boxplot-style summary for each variable.
<code>...</code>	Arguments passed on to <code>geom_line</code> .

Value

Concatenation of a `geom_line` value and optionally a `geom_ribbon` value.

See Also

Other geomes for `inla` and `inlabru` predictions: `gg.data.frame()`, `gg.matrix()`, `gg()`, `gm()`

Examples

```
if (bru_safe_inla() && require(ggplot2, quietly = TRUE)) {
  # Generate some data

  input.df <- data.frame(x = cos(1:10))
  input.df <- within(input.df, y <- 5 + 2 * cos(1:10) + rnorm(10, mean = 0, sd = 0.1))
}
```

```

# Fit a model with fixed effect 'x' and intercept 'Intercept'

fit <- bru(y ~ x, family = "gaussian", data = input.df)

# Predict posterior statistics of 'x'

xpost <- predict(fit, data = NULL, formula = ~x_latent)

# The statistics include mean, standard deviation, the 2.5% quantile, the median,
# the 97.5% quantile, minimum and maximum sample drawn from the posterior as well as
# the coefficient of variation and the variance.

xpost

# For a single variable like 'x' the default plotting method invoked by gg() will
# show these statistics in a fashion similar to a box plot:
ggplot() +
  gg(xpost)

# The predict function can also be used to simulataneously estimate posteriors
# of multiple variables:

xipost <- predict(fit,
  data = NULL,
  formula = ~ c(
    Intercept = Intercept_latent,
    x = x_latent
  )
)
xipost

# If we still want a plot in the previous style we have to set the bar parameter to TRUE

p1 <- ggplot() +
  gg(xipost, bar = TRUE)
p1

# Note that gg also understands the posterior estimates generated while running INLA

p2 <- ggplot() +
  gg(fit$summary.fixed, bar = TRUE)
multiplot(p1, p2)

# By default, if the prediction has more than one row, gg will plot the column 'mean' against
# the row index. This is for instance usefuul for predicting and plotting function
# but not very meaningful given the above example:

ggplot() +
  gg(xipost)

# For ease of use we can also type

```

```

plot(xipost)

# This type of plot will show a ribbon around the mean, which visualizes the upper and lower
# quantiles mentioned above (2.5 and 97.5%). Plotting the ribbon can be turned off using the
# \code{ribbon} parameter

ggplot() +
  gg(xipost, ribbon = FALSE)

# Much like the other geomes produced by gg we can adjust the plot using ggplot2 style
# commands, for instance

ggplot() +
  gg(xipost) +
  gg(xipost, mapping = aes(y = median), ribbon = FALSE, color = "red")
}

```

gg.RasterLayer

Geom for RasterLayer objects

Description

This function takes a RasterLayer object, converts it into a SpatialPixelsDataFrame and uses geom_tile to plot the data.

Usage

```

## S3 method for class 'RasterLayer'
gg(
  data,
  mapping = ggplot2::aes(x = .data[["x"]], y = .data[["y"]], fill = .data[["layer"]]),
  ...
)

```

Arguments

data	A RasterLayer object.
mapping	aesthetic mappings created by aes. These are passed on to geom_tile.
...	Arguments passed on to geom_tile.

Details

This function requires the raster and ggplot2 packages.

Value

An object returned by geom_tile

See Also

Other geomes for Raster data: [gg\(\)](#), [gm\(\)](#)

Examples

```
## Not run:
# Some features require the raster and spatstat.data packages.
if (require("spatstat.data", quietly = TRUE) &&
    require("raster", quietly = TRUE) &&
    require("ggplot2", quietly = TRUE)) {

  # Load Gorilla data
  data("gorillas", package = "spatstat.data")

  # Convert elevation covariate to RasterLayer

  elev <- as(gorillas.extra$elevation, "RasterLayer")

  # Plot the elevation

  ggplot() +
    gg(elev)
}

## End(Not run)
```

gg.SpatialGridDataFrame

Geom for SpatialGridDataFrame objects

Description

Coerces input `SpatialGridDataFrame` to `SpatialPixelsDataFrame` and calls [gg.SpatialPixelsDataFrame\(\)](#) to plot it. Requires the `ggplot2` package.

Usage

```
## S3 method for class 'SpatialGridDataFrame'
gg(data, ...)
```

Arguments

`data` A `SpatialGridDataFrame` object.
`...` Arguments passed on to [gg.SpatialPixelsDataFrame\(\)](#).

Value

A `geom_tile` value.

See Also

Other geomes for spatial data: [gg.SpatialLines\(\)](#), [gg.SpatialPixelsDataFrame\(\)](#), [gg.SpatialPixels\(\)](#), [gg.SpatialPoints\(\)](#), [gg.SpatialPolygons\(\)](#), [gg\(\)](#), [gm\(\)](#)

Examples

```
if (require(ggplot2, quietly = TRUE) &&
    require(ggpolypath, quietly = TRUE)) {
  # Load Gorilla data

  data("gorillas", package = "inlabru")

  # Plot Gorilla elevation covariate provided as SpatialPixelsDataFrame.
  # The same syntax applies to SpatialGridDataFrame objects.

  ggplot() +
    gg(gorillas$gcov$elevation)

  # Add Gorilla survey boundary and nest sightings

  ggplot() +
    gg(gorillas$gcov$elevation) +
    gg(gorillas$boundary) +
    gg(gorillas$nest)

  # Load pantropical dolphin data

  data("mexdolphins")

  # Plot the pantropical survey boundary, ship transects and dolphin sightings

  ggplot() +
    gg(mexdolphins$ppoly) + # survey boundary as SpatialPolygon
    gg(mexdolphins$samplers) + # ship transects as SpatialLines
    gg(mexdolphins$points) # dolphin sightings as SpatialPoints

  # Change color

  ggplot() +
    gg(mexdolphins$ppoly, color = "green") + # survey boundary as SpatialPolygon
    gg(mexdolphins$samplers, color = "red") + # ship transects as SpatialLines
    gg(mexdolphins$points, color = "blue") # dolphin sightings as SpatialPoints

  # Visualize data annotations: line width by segment number

  names(mexdolphins$samplers) # 'seg' holds the segment number
  ggplot() +
    gg(mexdolphins$samplers, aes(color = seg))

  # Visualize data annotations: point size by dolphin group size
```

```

names(mexdolphins$points) # 'size' holds the group size
ggplot() +
  gg(mexdolphins$points, aes(size = size))
}

```

gg.SpatialLines

Geom for SpatialLines objects

Description

Extracts start and end points of the lines and calls `geom_segment` to plot lines between them. Requires the `ggplot2` package.

Usage

```

## S3 method for class 'SpatialLines'
gg(data, mapping = NULL, crs = NULL, ...)

```

Arguments

<code>data</code>	A <code>SpatialLines</code> or <code>SpatialLinesDataFrame</code> object.
<code>mapping</code>	Aesthetic mappings created by <code>ggplot2::aes</code> or <code>ggplot2::aes_</code> used to update the default mapping. The default mapping is <code>ggplot2::aes(x = .data[[coordnames(data)[1]]], y = .data[[coordnames(data)[2]]], xend = .data[[paste0("end.", coordnames(data)[1])]], yend = .data[[paste0("end.", coordnames(data)[2])]])</code> .
<code>crs</code>	A CRS object defining the coordinate system to project the data to before plotting.
<code>...</code>	Arguments passed on to <code>ggplot2::geom_segment</code> .

Value

A ‘geom_segment’ return value.

See Also

Other geomes for spatial data: [gg.SpatialGridDataFrame\(\)](#), [gg.SpatialPixelsDataFrame\(\)](#), [gg.SpatialPixels\(\)](#), [gg.SpatialPoints\(\)](#), [gg.SpatialPolygons\(\)](#), [gg\(\)](#), [gm\(\)](#)

Examples

```

if (require(ggplot2, quietly = TRUE) &&
    require(ggpolypath, quietly = TRUE)) {
  # Load Gorilla data

  data("gorillas", package = "inlabru")

```

```

# Plot Gorilla elevation covariate provided as SpatialPixelsDataFrame.
# The same syntax applies to SpatialGridDataFrame objects.

ggplot() +
  gg(gorillas$gcov$elevation)

# Add Gorilla survey boundary and nest sightings

ggplot() +
  gg(gorillas$gcov$elevation) +
  gg(gorillas$boundary) +
  gg(gorillas$neests)

# Load pantropical dolphin data

data("mexdolphins")

# Plot the pantropical survey boundary, ship transects and dolphin sightings

ggplot() +
  gg(mexdolphins$ppoly) + # survey boundary as SpatialPolygon
  gg(mexdolphins$samplers) + # ship transects as SpatialLines
  gg(mexdolphins$points) # dolphin sightings as SpatialPoints

# Change color

ggplot() +
  gg(mexdolphins$ppoly, color = "green") + # survey boundary as SpatialPolygon
  gg(mexdolphins$samplers, color = "red") + # ship transects as SpatialLines
  gg(mexdolphins$points, color = "blue") # dolphin sightings as SpatialPoints

# Visualize data annotations: line width by segment number

names(mexdolphins$samplers) # 'seg' holds the segment number
ggplot() +
  gg(mexdolphins$samplers, aes(color = seg))

# Visualize data annotations: point size by dolphin group size

names(mexdolphins$points) # 'size' holds the group size
ggplot() +
  gg(mexdolphins$points, aes(size = size))
}

```

gg.SpatialPixels

Geom for SpatialPixels objects

Description

Uses `geom_point` to plot the pixel centers. Requires the `ggplot2` package.

Usage

```
## S3 method for class 'SpatialPixels'
gg(data, ...)
```

Arguments

```
data          A SpatialPixels object.
...           Arguments passed on to geom_tile.
```

Value

A geom_tile return value.

See Also

Other geomes for spatial data: [gg.SpatialGridDataFrame\(\)](#), [gg.SpatialLines\(\)](#), [gg.SpatialPixelsDataFrame\(\)](#), [gg.SpatialPoints\(\)](#), [gg.SpatialPolygons\(\)](#), [gg\(\)](#), [gm\(\)](#)

Examples

```
if (require("ggplot2", quietly = TRUE)) {
  # Load Gorilla data

  data(gorillas, package = "inlabru")

  # Turn elevation covariate into SpatialPixels
  pxl <- SpatialPixels(SpatialPoints(gorillas$gcov$elevation))

  # Plot the pixel centers
  ggplot() +
    gg(pxl, size = 0.1)
}
```

```
gg.SpatialPixelsDataFrame
```

Geom for SpatialPixelsDataFrame objects

Description

Coerces input SpatialPixelsDataFrame to data.frame and uses geom_tile to plot it. Requires the ggplot2 package.

Usage

```
## S3 method for class 'SpatialPixelsDataFrame'
gg(data, mapping = NULL, crs = NULL, mask = NULL, ...)
```


Arguments

<code>data</code>	A <code>SpatialPixelsDataFrame</code> object.
<code>mapping</code>	Aesthetic mappings created by <code>aes</code> used to update the default mapping. The default mapping is <code>ggplot2::aes(x = .data[[coordnames(data)[1]], y = .data[[coordnames(data)[2]], fill = .data[[names(data)[1]]]]]</code> .
<code>crs</code>	A CRS object defining the coordinate system to project the data to before plotting.
<code>mask</code>	A <code>SpatialPolygon</code> defining the region that is plotted.
<code>...</code>	Arguments passed on to <code>geom_tile</code> .

Value

A `geom_tile` return value.

See Also

Other geomes for spatial data: [gg.SpatialGridDataFrame\(\)](#), [gg.SpatialLines\(\)](#), [gg.SpatialPixels\(\)](#), [gg.SpatialPoints\(\)](#), [gg.SpatialPolygons\(\)](#), [gg\(\)](#), [gm\(\)](#)

Examples

```
if (require(ggplot2, quietly = TRUE) &&
    require(ggpolypath, quietly = TRUE)) {
  # Load Gorilla data

  data("gorillas", package = "inlabru")

  # Plot Gorilla elevation covariate provided as SpatialPixelsDataFrame.
  # The same syntax applies to SpatialGridDataFrame objects.

  ggplot() +
    gg(gorillas$gcov$elevation)

  # Add Gorilla survey boundary and nest sightings

  ggplot() +
    gg(gorillas$gcov$elevation) +
    gg(gorillas$boundary) +
    gg(gorillas$nest)

  # Load pantropical dolphin data

  data("mexdolphins")

  # Plot the pantropical survey boundary, ship transects and dolphin sightings

  ggplot() +
    gg(mexdolphins$ppoly) + # survey boundary as SpatialPolygon
    gg(mexdolphins$samplers) + # ship transects as SpatialLines
```

```

gg(mexdolphins$points) # dolphin sightings as SpatialPoints

# Change color

ggplot() +
  gg(mexdolphins$ppoly, color = "green") + # survey boundary as SpatialPolygon
  gg(mexdolphins$samplers, color = "red") + # ship transects as SpatialLines
  gg(mexdolphins$points, color = "blue") # dolphin sightings as SpatialPoints

# Visualize data annotations: line width by segment number

names(mexdolphins$samplers) # 'seg' holds the segment number
ggplot() +
  gg(mexdolphins$samplers, aes(color = seg))

# Visualize data annotations: point size by dolphin group size

names(mexdolphins$points) # 'size' holds the group size
ggplot() +
  gg(mexdolphins$points, aes(size = size))
}

```

gg.SpatialPoints

Geom for SpatialPoints objects

Description

This function coerces the `SpatialPoints` into a `data.frame` and uses `geom_point` to plot the points. Requires the `ggplot2` package.

Usage

```
## S3 method for class 'SpatialPoints'
gg(data, mapping = NULL, crs = NULL, ...)
```

Arguments

<code>data</code>	A <code>SpatialPoints</code> object.
<code>mapping</code>	Aesthetic mappings created by <code>aes</code> used to update the default mapping. The default mapping is <code>ggplot2::aes(x = .data[[coordnames(data)[1]], y = .data[[coordnames(data)[2]]])</code> .
<code>crs</code>	A CRS object defining the coordinate system to project the data to before plotting.
<code>...</code>	Arguments passed on to <code>geom_point</code> .

Value

A geom_point return value

See Also

Other geomes for spatial data: [gg.SpatialGridDataFrame\(\)](#), [gg.SpatialLines\(\)](#), [gg.SpatialPixelsDataFrame\(\)](#), [gg.SpatialPixels\(\)](#), [gg.SpatialPolygons\(\)](#), [gg\(\)](#), [gm\(\)](#)

Examples

```

if (require(ggplot2, quietly = TRUE) &&
    require(ggpolypath, quietly = TRUE)) {
  # Load Gorilla data

  data("gorillas", package = "inlabru")

  # Plot Gorilla elevation covariate provided as SpatialPixelsDataFrame.
  # The same syntax applies to SpatialGridDataFrame objects.

  ggplot() +
    gg(gorillas$gcov$elevation)

  # Add Gorilla survey boundary and nest sightings

  ggplot() +
    gg(gorillas$gcov$elevation) +
    gg(gorillas$boundary) +
    gg(gorillas$nest)

  # Load pantropical dolphin data

  data("mexdolphins")

  # Plot the pantropical survey boundary, ship transects and dolphin sightings

  ggplot() +
    gg(mexdolphins$ppoly) + # survey boundary as SpatialPolygon
    gg(mexdolphins$samplers) + # ship transects as SpatialLines
    gg(mexdolphins$points) # dolphin sightings as SpatialPoints

  # Change color

  ggplot() +
    gg(mexdolphins$ppoly, color = "green") + # survey boundary as SpatialPolygon
    gg(mexdolphins$samplers, color = "red") + # ship transects as SpatialLines
    gg(mexdolphins$points, color = "blue") # dolphin sightings as SpatialPoints

  # Visualize data annotations: line width by segment number

  names(mexdolphins$samplers) # 'seg' holds the segment number

```

```

ggplot() +
  gg(mexdolphinsamplers, aes(color = seg))

# Visualize data annotations: point size by dolphin group size

names(mexdolphinspoints) # 'size' holds the group size
ggplot() +
  gg(mexdolphinspoints, aes(size = size))
}

```

gg.SpatialPolygons *Geom for SpatialPolygons objects*

Description

Uses the `ggplot2::fortify()` function to turn the `SpatialPolygons` objects into a `data.frame`. Then calls `geom_polygon` to plot the polygons. Requires the `ggplot2` package.

Usage

```
## S3 method for class 'SpatialPolygons'
gg(data, mapping = NULL, crs = NULL, ...)
```

Arguments

<code>data</code>	A <code>SpatialPolygons</code> or <code>SpatialPolygonsDataFrame</code> object.
<code>mapping</code>	Aesthetic mappings created by <code>aes</code> used to update the default mapping. The default mapping is <code>ggplot2::aes(x = long, y = lat, group = group)</code> .
<code>crs</code>	A CRS object defining the coordinate system to project the data to before plotting.
<code>...</code>	Arguments passed on to <code>geom_polypath</code> . Unless specified by the user, the arguments <code>colour = "black"</code> (polygon colour) and <code>alpha = 0.2</code> (Alpha level for polygon filling).

Details

Requires the `ggpolypath` package to ensure proper plotting, since the `ggplot::geom_polygon` function doesn't always handle geometries with holes properly.

Value

A `ggpolypath::geom_polypath` object.

See Also

Other geomes for spatial data: [gg.SpatialGridDataFrame\(\)](#), [gg.SpatialLines\(\)](#), [gg.SpatialPixelsDataFrame\(\)](#), [gg.SpatialPixels\(\)](#), [gg.SpatialPoints\(\)](#), [gg\(\)](#), [gm\(\)](#)

Examples

```

if (require(ggplot2, quietly = TRUE) &&
    require(ggpolypath, quietly = TRUE)) {
  # Load Gorilla data

  data("gorillas", package = "inlabru")

  # Plot Gorilla elevation covariate provided as SpatialPixelsDataFrame.
  # The same syntax applies to SpatialGridDataFrame objects.

  ggplot() +
    gg(gorillas$gcov$elevation)

  # Add Gorilla survey boundary and nest sightings

  ggplot() +
    gg(gorillas$gcov$elevation) +
    gg(gorillas$boundary) +
    gg(gorillas$neests)

  # Load pantropical dolphin data

  data("mexdolphin")

  # Plot the pantropiical survey boundary, ship transects and dolphin sightings

  ggplot() +
    gg(mexdolphin$ppoly) + # survey boundary as SpatialPolygon
    gg(mexdolphin$samplers) + # ship transects as SpatialLines
    gg(mexdolphin$points) # dolphin sightings as SpatialPoints

  # Change color

  ggplot() +
    gg(mexdolphin$ppoly, color = "green") + # survey boundary as SpatialPolygon
    gg(mexdolphin$samplers, color = "red") + # ship transects as SpatialLines
    gg(mexdolphin$points, color = "blue") # dolphin sightings as SpatialPoints

  # Visualize data annotations: line width by segment number

  names(mexdolphin$samplers) # 'seg' holds the segment number
  ggplot() +
    gg(mexdolphin$samplers, aes(color = seg))

  # Visualize data annotations: point size by dolphin group size

  names(mexdolphin$points) # 'size' holds the group size
  ggplot() +
    gg(mexdolphin$points, aes(size = size))
}

```

globe

Plot a globe using rgl

Description

Creates a textured sphere and lon/lat coordinate annotations.

Usage

```
globe(  
  R = 1,  
  R.grid = 1.05,  
  specular = "black",  
  axes = FALSE,  
  box = FALSE,  
  xlab = "",  
  ylab = "",  
  zlab = ""  
)
```

Arguments

R	Radius of the globe
R.grid	Radius of the annotation sphere.
specular	Light color of specular effect.
axes	If TRUE, plot x, y and z axes.
box	If TRUE, plot a box around the globe.
xlab, ylab, zlab	Axes labels

Details

This function requires the `rgl` and `sphereplot` packages.

Value

No value, used for plotting side effect.

See Also

Other inlabru RGL tools: [glplot.SpatialLines\(\)](#), [glplot.SpatialPoints\(\)](#), [glplot.inla.mesh\(\)](#), [glplot\(\)](#)

Examples

```
## Not run:
if (bru_safe_inla() &&
    require("rgl", quietly = TRUE) &&
    require("sphereplot", quietly = TRUE)) {

  # Load pantropoical dolphin data

  data("mexdolphins", package = "inlabru")

  # Show the globe

  globe()

  # Add mesh, ship transects and dolphin sightings stored
  # as inla.mesh, SpatialLines and SpatialPoints objects, respectively

  glplot(mexdolphins$mesh)
  glplot(mexdolphins$samplers)
  glplot(mexdolphins$points)
}

## End(Not run)
```

glplot

Render Spatial and inla.mesh objects using RGL*

Description

glplot is a generic function for renders various kinds of spatial objects, i.e. Spatial* data and inla.mesh objects. The function invokes particular methods which depend on the class of the first argument.

Usage

```
glplot(object, ...)
```

Arguments

object	an object used to select a method.
...	further arguments passed to or from other methods.

See Also

Other inlabru RGL tools: [globe\(\)](#), [glplot.SpatialLines\(\)](#), [glplot.SpatialPoints\(\)](#), [glplot.inla.mesh\(\)](#)

Examples

```
## Not run:
if (bru_safe_inla() &&
    require("rgl", quietly = TRUE) &&
    require("sphereplot", quietly = TRUE)) {

  # Load pantropoical dolphin data

  data("mexdolphins", package = "inlabru")

  # Show the globe

  globe()

  # Add mesh, ship transects and dolphin sightings stored
  # as inla.mesh, SpatialLines and SpatialPoints objects, respectively

  glplot(mexdolphins$mesh)
  glplot(mexdolphins$samplers)
  glplot(mexdolphins$points)
}

## End(Not run)
```

glplot.inla.mesh

Visualize SpatialPoints using RGL

Description

This function transforms the mesh to 3D cartesian coordinates and uses `inla.plot.mesh()` with `rgl=TRUE` to plot the result.

Usage

```
## S3 method for class 'inla.mesh'
glplot(object, add = TRUE, col = NULL, ...)
```

Arguments

<code>object</code>	an <code>inla.mesh</code> object.
<code>add</code>	If <code>TRUE</code> , add the lines to an existing plot. If <code>FALSE</code> , create new plot.
<code>col</code>	Color specification. A single named color, a vector of scalar values, or a matrix of RGB values.
<code>...</code>	Parameters passed on to <code>plot.inla.mesh()</code>

See Also

Other `inlabru` RGL tools: [globe\(\)](#), [glplot.SpatialLines\(\)](#), [glplot.SpatialPoints\(\)](#), [glplot\(\)](#)

Examples

```
## Not run:
if (bru_safe_inla() &&
    require("rgl", quietly = TRUE) &&
    require("sphereplot", quietly = TRUE)) {

  # Load pantropoical dolphin data

  data("mexdolphins", package = "inlabru")

  # Show the globe

  globe()

  # Add mesh, ship transects and dolphin sightings stored
  # as inla.mesh, SpatialLines and SpatialPoints objects, respectively

  glplot(mexdolphins$mesh)
  glplot(mexdolphins$samplers)
  glplot(mexdolphins$points)
}

## End(Not run)
```

glplot.SpatialLines *Visualize SpatialLines using RGL*

Description

This function will calculate a cartesian representation of the lines provided and use `rgl.linestrip()` in order to render them.

Usage

```
## S3 method for class 'SpatialLines'
glplot(object, add = TRUE, ...)
```

Arguments

<code>object</code>	a <code>SpatialLines</code> or <code>SpatialLinesDataFrame</code> object.
<code>add</code>	If <code>TRUE</code> , add the lines to an existing plot. If <code>FALSE</code> , create new plot.
<code>...</code>	Parameters passed on to <code>rgl.linestrips()</code> .

See Also

Other inlabru RGL tools: [globe\(\)](#), [glplot.SpatialPoints\(\)](#), [glplot.inla.mesh\(\)](#), [glplot\(\)](#)

Examples

```
## Not run:
if (bru_safe_inla() &&
    require("rgl", quietly = TRUE) &&
    require("sphereplot", quietly = TRUE)) {

  # Load pantropoical dolphin data

  data("mexdolphins", package = "inlabru")

  # Show the globe

  globe()

  # Add mesh, ship transects and dolphin sightings stored
  # as inla.mesh, SpatialLines and SpatialPoints objects, respectively

  glplot(mexdolphins$mesh)
  glplot(mexdolphins$samplers)
  glplot(mexdolphins$points)
}

## End(Not run)
```

glplot.SpatialPoints *Visualize SpatialPoints using RGL*

Description

This function will calculate the cartesian coordinates of the points provided and use `rgl.points()` in order to render them.

Usage

```
## S3 method for class 'SpatialPoints'
glplot(object, add = TRUE, color = "red", ...)
```

Arguments

<code>object</code>	a <code>SpatialPoints</code> or <code>SpatialPointsDataFrame</code> object.
<code>add</code>	If <code>TRUE</code> , add the points to an existing plot. If <code>FALSE</code> , create new plot.
<code>color</code>	vector of R color characters. See <code>rgl.material()</code> for details.
<code>...</code>	Parameters passed on to <code>rgl.points()</code>

See Also

Other inlabru RGL tools: [globe\(\)](#), [glplot.SpatialLines\(\)](#), [glplot.inla.mesh\(\)](#), [glplot\(\)](#)

Examples

```
## Not run:
if (bru_safe_inla() &&
    require("rgl", quietly = TRUE) &&
    require("sphereplot", quietly = TRUE)) {

  # Load pantropoical dolphin data

  data("mexdolphins", package = "inlabru")

  # Show the globe

  globe()

  # Add mesh, ship transects and dolphin sightings stored
  # as inla.mesh, SpatialLines and SpatialPoints objects, respectively

  glplot(mexdolphins$mesh)
  glplot(mexdolphins$samplers)
  glplot(mexdolphins$points)
}

## End(Not run)
```

gm *ggplot geom for spatial data*

Description

gm is a wrapper for the [gg](#) method. It will take the first argument and transform its coordinate system to latitude and longitude. Thereafter, [gg](#) is called using the transformed data and the arguments provided via `...`. gm is intended to replace gg whenever the data is supposed to be plotted over a spatial map generated by [gmap](#), which only works if the coordinate system is latitude/longitude.

Usage

```
gm(data, ...)
```

Arguments

data an object for which to generate a geom.
 ... Arguments passed on to [gg\(\)](#).

Value

The form of the value returned by gm depends on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

See Also

Other geomes for inla and inlabru predictions: `gg.data.frame()`, `gg.matrix()`, `gg.prediction()`, `gg()`

Other geomes for spatial data: `gg.SpatialGridDataFrame()`, `gg.SpatialLines()`, `gg.SpatialPixelsDataFrame()`, `gg.SpatialPixels()`, `gg.SpatialPoints()`, `gg.SpatialPolygons()`, `gg()`

Other geomes for meshes: `gg.inla.mesh.1d()`, `gg.inla.mesh()`, `gg()`

Other geomes for Raster data: `gg.RasterLayer()`, `gg()`

Examples

```
## Not run:
if (require("ggplot2", quietly = TRUE) &&
    require(ggpolypath, quietly = TRUE)) {
  # Load the Gorilla data
  data(gorillas, package = "inlabru")

  # Create a base map centered around the nests and plot the boundary as well as the nests
  gmap(gorillas$nests, matype = "satellite") +
    gm(gorillas$boundary) +
    gm(gorillas$nests, color = "white", size = 0.5)
}

## End(Not run)
```

`gmap`

Plot a map using extend of a spatial object

Description

Uses `get_map()` to query map services like Google Maps for a region centered around the spatial object provided. Then calls `gmap()` to plot the map.

Usage

```
gmap(data, ...)
```

Arguments

`data` A Spatial* object.
`...` Arguments passed on to `get_map()`.

Details

This function requires the `ggmap` package.

Value

a ggplot object

Examples

```
## Not run:
if (require("ggplot2", quietly = TRUE) &&
    require(ggpolypath, quietly = TRUE)) {
  # Load the Gorilla data
  data(gorillas, package = "inlabru")

  # Create a base map centered around the nests and plot the boundary as well
  # as the nests
  ggplot() +
    gg(gorillas$boundary) +
    gg(gorillas$nests, color = "white", size = 0.5)
  if (requireNamespace("ggmap", quietly = TRUE)) {
    gmap(gorillas$nests, matype = "satellite") +
      gm(gorillas$boundary) +
      gm(gorillas$nests, color = "white", size = 0.5)
  }
}

## End(Not run)
```

gorillas

Gorilla nesting sites

Description

This is the gorillas dataset from the package spatstat.data, reformatted as point process data for use with inlabru.

Usage

```
data(gorillas)
```

Format

The data are a list that contains these elements:

nests: A SpatialPointsDataFrame object containing the locations of the gorilla nests.

boundary: An SpatialPolygonsDataFrame object defining the boundary of the region that was searched for the nests.

mesh: An inla.mesh object containing a mesh that can be used with function lgcp to fit a LGCP to the nest data.

gcov: A list of SpatialGridDataFrame objects, one for each of these spatial covariates:

aspect Compass direction of the terrain slope. Categorical, with levels N, NE, E, SE, S, SW, W and NW, which are coded as integers 1 to 8.

elevation Digital elevation of terrain, in metres.

heat Heat Load Index at each point on the surface (Beer's aspect), discretised. Categorical with values Warmest (Beer's aspect between 0 and 0.999), Moderate (Beer's aspect between 1 and 1.999), Coolest (Beer's aspect equals 2). These are coded as integers 1, 2 and 3, in that order.

slopangle Terrain slope, in degrees.

sloptype Type of slope. Categorical, with values Valley, Toe (toe slope), Flat, Midslope, Upper and Ridge. These are coded as integers 1 to 6.

vegetation Vegetation type: a categorical variable with 6 levels coded as integers 1 to 6 (in order of increasing expected habitat suitability)

waterdist Euclidean distance from nearest water body, in metres.

plotsample Plot sample of gorilla nests, sampling 9x9 over the region, with 60\

counts A SpatialPointsDataFrame frame with elements x, y, count, exposure, being the x- and y-coordinates of the centre of each plot, the count in each plot and the area of each plot.

plots A SpatialPolygonsDataFrame defining the individual plot boundaries.

nests A SpatialPointsDataFrame giving the locations of each detected nest.

Source

Library spatstat.data.

References

Funwi-Gabga, N. (2008) A pastoralist survey and fire impact assessment in the Kagwene Gorilla Sanctuary, Cameroon. M.Sc. thesis, Geology and Environmental Science, University of Buea, Cameroon.

Funwi-Gabga, N. and Mateu, J. (2012) Understanding the nesting spatial behaviour of gorillas in the Kagwene Sanctuary, Cameroon. Stochastic Environmental Research and Risk Assessment 26 (6), 793-811.

Examples

```
if (require(ggplot2, quietly = TRUE)) {
  data(gorillas, package = "inlabru") # get the data
  # extract all the objects, for convenience:

  # plot all the nests, mesh and boundary
  ggplot() +
    gg(gorillas$mesh) +
    gg(gorillas$boundary) +
    gg(gorillas$nests)

  # Plot the elevation covariate
  plot(gorillas$gcov$elevation)
```

```

# Plot the plot sample
ggplot() +
  gg(gorillas$plotsample$plots) +
  gg(gorillas$plotsample$nest)
}

```

inla.stack.mjoin

Join stacks intended to be run with different likelihoods

Description

Join stacks intended to be run with different likelihoods

Usage

```

inla.stack.mjoin(
  ...,
  compress = TRUE,
  remove.unused = TRUE,
  old.names = "BRU.response",
  new.name = "BRU.response"
)

```

Arguments

...	List of stacks that contain vector observations (existing multi-likelihood observation matrices are also permitted)
compress	If TRUE, compress the model by removing duplicated rows of effects, replacing the corresponding A-matrix columns with a single column containing the sum.
remove.unused	If TRUE, compress the model by removing rows of effects corresponding to all-zero columns in the A matrix (and removing those columns).
old.names	A vector of strings with the names of the observation vector/matrix for each stack. If a single string, this is assumed for all the stacks. (default "BRU.response")
new.name	The name to be used for the expanded observation matrix, possibly the same as an old name. (default "BRU.response")

 inlabru

inlabru

Description

Convenient model fitting using (iterated) INLA.

Details

inlabru facilitates Bayesian spatial modelling using integrated nested Laplace approximations. It is heavily based on R-inla (<https://www.r-inla.org>) but adds additional modelling abilities and simplified syntax for (in particular) spatial models. Tutorials and more information can be found at <https://inlabru-org.github.io/inlabru/> and <http://www.inlabru.org/>. The iterative method used for non-linear predictors is documented in the method vignette.

The main function for inference using inlabru is `bru()`. For point process inference `lgcp()` is a good starting point. The general model specification details is documented in `component()` and `like()`. Posterior quantities beyond the basic summaries can be calculated with a `predict()` method, documented in `predict.bru()`.

The package comes with multiple real world data sets, namely `gorillas`, `mexdolphins`, `seals`. Plotting these data sets is straight forward using inlabru's extensions to `ggplot2`, e.g. the `gg()` function. For educational purposes some simulated data sets are available as well, e.g. `Poisson1_1D`, `Poisson2_1D`, `Poisson2_1D` and `toygroups`.

Author(s)

Fabian E. Bachl <bachlfab@gmail.com> and Finn Lindgren <finn.lindgren@gmail.com>

 int

Weighted summation (integration) of data frame subsets

Description

A typical task in statistical inference is to integrate a (multivariate) function along one or more dimensions of its domain. For this purpose, the function is evaluated at some points in the domain and the values are summed up using weights that depend on the area being integrated over. This function performs the weighting and summation conditional for each level of the dimensions that are not integrated over. The parameter `dims` states the dimensions to integrate over. The set of dimensions that are held fixed is the set difference of all column names in `data` and the dimensions stated by `dims`.

Usage

```
int(data, values, dims = NULL)
```


Arguments

data	A data.frame or Spatial object. Has to have a weight column with numeric values.
values	Numerical values to be summed up, usually the result of function evaluations.
dims	Column names (dimension names) of the data object to integrate over.

Value

A data.frame of integrals, one for each level of the cross product of all dimensions not being integrated over.

Examples

```
# ipoints needs INLA
if (bru_safe_inla(quietly = TRUE)) {
  # Create integration points in two dimensions, x and y

  ips <- cprod(
    ipoints(c(0, 10), 10, name = "x"),
    ipoints(c(1, 5), 10, name = "y")
  )

  # The sizes of the domains are 10 and 4 for x and y, respectively.
  # Integrating f(x,y) = 1 along x and y should result in the total
  # domain size 40

  int(ips, rep(1, nrow(ips)), c("x", "y"))
}
```

ipoints

Generate integration points

Description

This function generates points in one or two dimensions with a weight attached to each point. The weighted sum of a function evaluated at these points is the integral of that function approximated by linear basis functions. The parameter `samplers` describes the area(s) integrated over.

In case of a single dimension `samplers` is supposed to be a two-column matrix where each row describes the start and end point of the interval to integrate over. In the two-dimensional case `samplers` can be either a `SpatialPolygon`, an `inla.mesh` or a `SpatialLinesDataFrame` describing the area to integrate over. If a `SpatialLineDataFrame` is provided it has to have a column called 'weight' in order to indicate the width of the line.

The domain parameter is an `inla.mesh.1d` or `inla.mesh` object that can be employed to project the integration points to the vertices of the mesh. This reduces the final number of integration points

and reduces the computational cost of the integration. The projection can also prevent numerical issues in spatial LGCP models where each observed point is ideally surrounded by three integration point sitting at the corresponding mesh vertices. This is controlled by `int. args$method="stable"` (default) or `"direct"`, where the latter uses the integration points directly, without aggregating to the mesh vertices.

For convenience, the `domain` parameter can also be a single integer setting the number of equally spaced integration points in the one-dimensional case.

Usage

```
ipoints(
  samplers = NULL,
  domain = NULL,
  name = NULL,
  group = NULL,
  int.args = NULL,
  project = NULL
)
```

Arguments

<code>samplers</code>	Description of the integration region boundary. In 1D, a length 2 vector or two-column matrix where each row describes an interval, or NULL. In 2D either a <code>SpatialPolygon</code> or a <code>SpatialLinesDataFrame</code> with a weight column defining the width of the a transect line, and optionally further columns used by the <code>group</code> argument, or NULL. When <code>domain</code> is NULL, <code>samplers</code> may also be an <code>inla.mesh.1d</code> or <code>inla.mesh</code> object, that is then treated as a <code>domain</code> argument instead.
<code>domain</code>	Either <ul style="list-style-type: none"> • when <code>samplers</code> is a 1D interval(s) definition only, <code>domain</code> can be a single integer for the number of integration points to place in each 1D interval, overriding <code>int.args[["nsub1"]]</code>, and otherwise • when <code>samplers</code> is NULL, <code>domain</code> can be a numeric vector of points, each given integration weight 1 (and no additional points are added in between), • an <code>inla.mesh.1d</code> object for continuous 1D integration, or • an <code>inla.mesh.2d</code> object for continuous 2D integration.
<code>name</code>	Character array stating the name of the domains dimension(s). If NULL, the names are taken from coordinate names from <code>samplers</code> for <code>Spatial*</code> objects, otherwise "x", "y", "coordinateZ" for 2D regions and "x" for 1D regions
<code>group</code>	Column names of the <code>samplers</code> object (if applicable) for which the integration points are calculated independently and not merged when aggregating to mesh nodes.
<code>int.args</code>	List of arguments passed to <code>bru_int_polygon</code> . <ul style="list-style-type: none"> • <code>method</code>: "stable" (to aggregate integration weights onto mesh nodes) or "direct" (to construct a within triangle/segment integration scheme without aggregating onto mesh nodes)

- `nsub1`, `nsub2`: integers controlling the number of internal integration points before aggregation. Points per triangle: $(nsub2+1)^2$. Points per knot segment: `nsub1`
 - `poly_method`: if set to "legacy", selects an old polygon integration method that doesn't handle holes. Currently only used for debugging purposes.
- `project` Deprecated in favour of `int.args(method=...)`. If TRUE, aggregate the integration points to mesh vertices. Default: `project = (identical(int.args$method, "stable"))`

Value

A data.frame or SpatialPointsDataFrame of 1D and 2D integration points, respectively.

Author(s)

Fabian E. Bachl <bachlfab@gmail.com> and <finn.lindgren@gmail.com>

Examples

```
if (require("INLA", quietly = TRUE) &&
    require("ggplot2", quietly = TRUE)) {

  # Create 50 integration points covering the dimension 'myDim' between 0 and 10.

  ips <- ipoints(c(0, 10), 50, name = "myDim")
  plot(ips)

  # Create integration points for the two intervals [0,3] and [5,10]

  ips <- ipoints(matrix(c(0, 3, 5, 10), nrow = 2, byrow = TRUE), 50)
  plot(ips)

  # Convert a 1D mesh into integration points
  mesh <- inla.mesh.1d(seq(0, 10, by = 1))
  ips <- ipoints(mesh, name = "time")
  plot(ips)

  # Obtain 2D integration points from a SpatialPolygon

  data(gorillas, package = "inlabru")
  ips <- ipoints(gorillas$boundary)
  ggplot() +
    gg(gorillas$boundary) +
    gg(ips, aes(size = weight))

  #' Project integration points to mesh vertices

  ips <- ipoints(gorillas$boundary, domain = gorillas$mesh)
```

```

ggplot() +
  gg(gorillas$mesh) +
  gg(gorillas$boundary) +
  gg(ips, aes(size = weight))

# Turn a 2D mesh into integration points

ips <- ipoints(gorillas$mesh)
ggplot() +
  gg(gorillas$boundary) +
  gg(ips, aes(size = weight))
}

```

lgcp

Log Gaussian Cox process (LGCP) inference using INLA

Description

This function performs inference on a LGCP observed via points residing possibly multiple dimensions. These dimensions are defined via the left hand side of the formula provided via the model parameter. The left hand side determines the intensity function that is assumed to drive the LGCP. This may include effects that lead to a thinning (filtering) of the point process. By default, the log intensity is assumed to be a linear combination of the effects defined by the formula's RHS. More sophisticated models, e.g. non-linear thinning, can be achieved by using the predictor argument. The latter requires multiple runs of INLA for improving the required approximation of the predictor. In many applications the LGCP is only observed through subsets of the dimensions the process is living in. For example, spatial point realizations may only be known in sub-areas of the modelled space. These observed subsets of the LGCP domain are called samplers and can be provided via the respective parameter. If samplers is NULL it is assumed that all of the LGCP's dimensions have been observed completely.

Usage

```

lgcp(
  components,
  data,
  samplers = NULL,
  domain = NULL,
  ips = NULL,
  formula = . ~ .,
  E = NULL,
  ...,
  options = list()
)

```

Arguments

components	A formula describing the latent components
data	A data frame or <code>SpatialPoints(DataFrame)</code> object
samplers	A data frame or <code>Spatial[Points/Lines/Polygons]DataFrame</code> objects
domain	Named list of domain definitions
ips	Integration points (overrides samplers)
formula	If <code>NULL</code> , the linear combination implied by the components is used as a predictor for the point location intensity. If a (possibly non-linear) expression is provided the respective Taylor approximation is used as a predictor. Multiple runs if INLA are then required for a better approximation of the posterior.
E	Single numeric used rescale all integration weights by a fixed factor
...	Further arguments passed on to <code>like()</code>
options	See <code>bru_options_set()</code>

Value

An `bru()` object

Examples

```
if (bru_safe_inla() &&
    require(ggplot2, quietly = TRUE)) {

  # Load the Gorilla data
  data(gorillas, package = "inlabru")

  # Plot the Gorilla nests, the mesh and the survey boundary
  ggplot() +
    gg(gorillas$mesh) +
    gg(gorillas$nests) +
    gg(gorillas$boundary) +
    coord_fixed()

  # Define SPDE prior
  matern <- INLA::inla.spde2.pcmatern(gorillas$mesh,
    prior.sigma = c(0.1, 0.01),
    prior.range = c(0.01, 0.01)
  )

  # Define domain of the LGCP as well as the model components (spatial SPDE
  # effect and Intercept)
  cmp <- coordinates ~ mySmooth(coordinates, model = matern) + Intercept(1)

  # Fit the model (with int.strategy="eb" to make the example take less time)
  fit <- lgcp(cmp, gorillas$nests,
    samplers = gorillas$boundary,
    domain = list(coordinates = gorillas$mesh),
```

```

    options = list(control.inla = list(int.strategy = "eb"))
  )

  # Predict the spatial intensity surface
  lambda <- predict(fit, pixels(gorillas$mesh), ~ exp(mySmooth + Intercept))

  # Plot the intensity
  ggplot() +
    gg(lambda) +
    gg(gorillas$mesh) +
    gg(gorillas$nest) +
    gg(gorillas$boundary) +
    coord_fixed()
}

```

like

Likelihood construction for usage with [bru\(\)](#)

Description

Likelihood construction for usage with [bru\(\)](#)

Usage

```

like(
  formula = . ~ .,
  family = "gaussian",
  data = NULL,
  response_data = NULL,
  mesh = NULL,
  E = NULL,
  Ntrials = NULL,
  samplers = NULL,
  ips = NULL,
  domain = NULL,
  include = NULL,
  exclude = NULL,
  allow_latent = FALSE,
  allow_combine = NULL,
  control.family = NULL,
  options = list(),
  .envir = parent.frame()
)

like_list(...)

```

```
## S3 method for class 'list'
like_list(object, envir = NULL, ...)

## S3 method for class 'bru_like'
like_list(..., envir = NULL)

## S3 method for class 'bru_like_list'
x[i]
```

Arguments

formula	a formula where the right hand side is a general R expression defines the predictor used in the model.
family	A string identifying a valid INLA::inla likelihood family. The default is gaussian with identity link. In addition to the likelihoods provided by inla (see names(INLA::inla.models())\$lik inlabru supports fitting latent Gaussian Cox processes via family = "cp". As an alternative to bru(), the lgcp() function provides a convenient interface to fitting Cox processes.
data	Likelihood-specific data, as a data.frame or SpatialPoints[DataFrame] object.
response_data	Likelihood-specific data for models that need different size/format for inputs and response variables, as a data.frame or SpatialPoints[DataFrame] object.
mesh	An inla.mesh object. Obsolete.
E	Exposure parameter for family = 'poisson' passed on to INLA::inla. Special case if family is 'cp': rescale all integration weights by E. Default taken from options\$E.
Ntrials	A vector containing the number of trials for the 'binomial' likelihood. Default value is rep(1, n.data). Default taken from options\$Ntrials.
samplers	Integration domain for 'cp' family.
ips	Integration points for 'cp' family. Overrides samplers.
domain	Named list of domain definitions.
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)
allow_latent	logical. If TRUE, the latent state of each component is directly available to the predictor expression, with a _latent suffix. This also makes evaluator functions with suffix _eval available, taking parameters main, group, and replicate, taking values for where to evaluate the component effect that are different than those defined in the component definition itself (see component_eval()). Default FALSE
allow_combine	logical; If TRUE, the predictor expression may involve several rows of the input data to influence the same row. Default FALSE, but forced to TRUE if response_data is NULL or data is a list

<code>control.family</code>	A optional list of <code>INLA::control.family</code> options
<code>options</code>	A <code>bru_options</code> options object or a list of options passed on to <code>bru_options()</code>
<code>.envir</code>	The evaluation environment to use for special arguments (E and Ntrials) if not found in <code>response_data</code> or <code>data</code> . Defaults to the calling environment.
<code>...</code>	For <code>like_list.bru_like</code> , one or more <code>bru_like</code> objects
<code>object</code>	A list of <code>bru_like</code> objects
<code>envir</code>	An optional environment for the new <code>bru_like_list</code> object
<code>x</code>	<code>bru_like_list</code> object from which to extract element(s)
<code>i</code>	indices specifying elements to extract

Details

- `like_list`: Combine a `bru_like` likelihoods into a `bru_like_list` object
- `like_list.list`: Combine a list of `bru_like` likelihoods into a `bru_like_list` object
- `like_list.bru_like`: Combine several `bru_like` likelihoods into a `bru_like_list` object

Value

A likelihood configuration which can be used to parameterize `bru()`.

Author(s)

Fabian E. Bachl <bachlfab@gmail.com>

Examples

```
if (bru_safe_inla() &&
    require(ggplot2, quietly = TRUE)) {

  # The like function's main purpose is to set up models with multiple likelihoods.
  # The following example generates some random covariates which are observed through
  # two different random effect models with different likelihoods

  # Generate the data

  set.seed(123)

  n1 <- 200
  n2 <- 10

  x1 <- runif(n1)
  x2 <- runif(n2)
  z2 <- runif(n2)

  y1 <- rnorm(n1, mean = 2 * x1 + 3)
  y2 <- rpois(n2, lambda = exp(2 * x2 + z2 + 3))
}
```



```

df1 <- data.frame(y = y1, x = x1)
df2 <- data.frame(y = y2, x = x2, z = z2)

# Single likelihood models and inference using bru are done via

cmp1 <- y ~ -1 + Intercept(1) + x
fit1 <- bru(cmp1, family = "gaussian", data = df1)
summary(fit1)

cmp2 <- y ~ -1 + Intercept(1) + x + z
fit2 <- bru(cmp2, family = "poisson", data = df2)
summary(fit2)

# A joint model has two likelihoods, which are set up using the like function

lik1 <- like("gaussian", formula = y ~ x + Intercept, data = df1)
lik2 <- like("poisson", formula = y ~ x + z + Intercept, data = df2)

# The union of effects of both models gives the components needed to run bru

jcmp <- ~ x + z + Intercept(1)
jfit <- bru(jcmp, lik1, lik2)

# Compare the estimates

p1 <- ggplot() +
  gg(fit1$summary.fixed, bar = TRUE) +
  ylim(0, 4) +
  ggtitle("Model 1")
p2 <- ggplot() +
  gg(fit2$summary.fixed, bar = TRUE) +
  ylim(0, 4) +
  ggtitle("Model 2")
pj <- ggplot() +
  gg(jfit$summary.fixed, bar = TRUE) +
  ylim(0, 4) +
  ggtitle("Joint model")

multiplot(p1, p2, pj)
}

```

Description

This is a version of the mexdolphins dataset from the package `dsm`, reformatted as point process data for use with `inlabru`. The data are from a combination of several NOAA shipboard surveys

conducted on pan-tropical spotted dolphins in the Gulf of Mexico. 47 observations of groups of dolphins were detected. The group size was recorded, as well as the Beaufort sea state at the time of the observation. Transect width is 16 km, i.e. maximal detection distance 8 km (transect half-width 8 km).

Usage

```
data(mexdolphins)
```

Format

A list of objects:

points: A `SpatialPointsDataFrame` object containing the locations of detected dolphin groups, with their size as an attribute.

samplers: A `SpatialLinesDataFrame` object containing the transect lines that were surveyed.

mesh: An `inla.mesh` object containing a Delaunay triangulation mesh (a type of discretization of continuous space) covering the survey region.

ppoly: An `SpatialPolygonsDataFrame` object defining the boundary of the survey region.

simulated: A `SpatialPointsDataFrame` object containing the locations of a *simulated* population of dolphin groups. The population was simulated from a 'codeinlabru' model fitted to the actual survey data. Note that the simulated data do not have any associated size information.

Source

Library `dsm`.

References

Halpin, P.N., A.J. Read, E. Fujioka, B.D. Best, B. Donnelly, L.J. Hazen, C. Kot, K. Urian, E. LaBrecque, A. Dimatteo, J. Cleary, C. Good, L.B. Crowder, and K.D. Hyrenbach. 2009. OBIS-SEAMAP: The world data center for marine mammal, sea bird, and sea turtle distributions. *Oceanography* 22(2):104-115

NOAA Southeast Fisheries Science Center. 1996. Report of a Cetacean Survey of Oceanic and Selected Continental Shelf Waters of the Northern Gulf of Mexico aboard NOAA Ship Oregon II (Cruise 220)

Examples

```
if (bru_safe_inla(quietly = TRUE) &&
    require("ggplot2", quietly = TRUE)) {
  data(mexdolphins, package = "inlabru")
  ggplot() +
    gg(mexdolphins$mesh) +
    gg(mexdolphins$ppoly, color = "blue") +
    gg(mexdolphins$samplers) +
    gg(mexdolphins$points, aes(size = size), color = "red") +
    coord_equal()
```

```

  ggplot() +
    gg(mexdolphins$mesh, col = mexdolphins$lambda, mask = mexdolphins$ppoly) +
    coord_equal()
}

## Not run:
if (requireNamespace("ggmap", quietly = TRUE) &&
    require("ggplot2", quietly = TRUE)) {
  gmap(mexdolphins$depth) +
    gm(mexdolphins$ppoly, color = "blue") +
    gm(mexdolphins$samplers) +
    gm(mexdolphins$points, aes(size = size), color = "red")

  gmap(mexdolphins$depth) +
    gm(mexdolphins$depth, aes(col = depth)) +
    gm(mexdolphins$ppoly)
}

## End(Not run)

```

 mrsea

Marine renewables strategic environmental assessment

Description

Data imported from package MRSea, see <http://creem2.st-andrews.ac.uk/software/>

Usage

```
data(mrsea)
```

Format

A list of objects:

points: A SpatialPointsDataFrame object containing the locations of XXXXX.

samplers: A SpatialLinesDataFrame object containing the transect lines that were surveyed.

mesh: An `inla.mesh` object containing a Delaunay triangulation mesh (a type of discretization of continuous space) covering the survey region.

boundary: An SpatialPolygonsDataFrame object defining the boundary of the survey region.

cover: An SpatialPointsDataFrame containing sea depth estimates.

Source

Library MRSea.

References

NONE YET

Examples

```
if (require(ggplot2, quietly = TRUE)) {  
  data(mrsea)  
  ggplot() +  
    gg(mrsea$mesh) +  
    gg(mrsea$samplers) +  
    gg(mrsea$points) +  
    gg(mrsea$boundary)  
}
```

multiplot

Multiple ggplots on a page.

Description

Renders multiple ggplots on a single page.

Usage

```
multiplot(..., plotlist = NULL, cols = 1, layout = NULL)
```

Arguments

...	Comma-separated ggplot objects.
plotlist	A list of ggplot objects - an alternative to the comma-separated argument above.
cols	Number of columns of plots on the page.
layout	A matrix specifying the layout. If present, 'cols' is ignored. If the layout is something like <code>matrix(c(1,2,3,3), nrow=2, byrow=TRUE)</code> , then plot 1 will go in the upper left, 2 will go in the upper right, and 3 will go all the way across the bottom.

Author(s)

David L. Borchers <dlb@st-andrews.ac.uk>

Source

[http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_\(ggplot2\)/](http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)/)

Examples

```

if (require("ggplot2", quietly = TRUE)) {
  df <- data.frame(x = 1:10, y = 1:10, z = 11:20)
  p1 <- ggplot(data = df) +
    geom_line(mapping = aes(x, y), color = "red")
  p2 <- ggplot(data = df) +
    geom_line(mapping = aes(x, z), color = "blue")
  multiplot(p1, p2, cols = 2)
}

```

pixels

Generate SpatialPixels covering an inla.mesh

Description

Generate SpatialPixels covering an inla.mesh

Usage

```
pixels(mesh, nx = 150, ny = 150, mask = TRUE)
```

Arguments

mesh	An inla.mesh object
nx	Number of pixels in x direction
ny	Number of pixels in y direction
mask	If logical and TRUE, remove pixels that are outside the mesh. If mask is a Spatial object, only return pixels covered by this object.

Value

SpatialPixelsDataFrame covering the mesh

Author(s)

Fabian E. Bachl <bachlfab@gmail.com>

Examples

```

if (require(ggplot2, quietly = TRUE)) {
  data("mrsea", package = "inlabru")
  pxl <- pixels(mrsea$mesh, nx = 50, ny = 50, mask = mrsea$boundary)
  ggplot() +
    gg(pxl, fill = "grey", alpha = 0.5) +
    gg(mrsea$mesh)
}

```

`plot.bru`*Plot method for posterior marginals estimated by bru*

Description

`bru()` uses `INLA::inla()` to fit models. The latter estimates the posterior densities of all random effects in the model. This function serves to access and plot the posterior densities in a convenient way.

Requires the `ggplot2` package.

Usage

```
## S3 method for class 'bru'
plot(x, ...)
```

Arguments

<code>x</code>	a fitted <code>bru()</code> model.
<code>...</code>	A character naming the effect to plot, e.g. "Intercept". For random effects, adding <code>index = ...</code> plots the density for a single component of the latent model.

Value

an object of class `gg`

Examples

```
## Not run:
if (require("ggplot2", quietly = TRUE)) {
  # Generate some data and fit a simple model
  input.df <- data.frame(x = cos(1:10))
  input.df <- within(input.df, y <- 5 + 2 * cos(1:10) + rnorm(10, mean = 0, sd = 0.1))
  fit <- bru(y ~ x, family = "gaussian", data = input.df)
  summary(fit)

  # Plot the posterior of the model's x-effect
  plot(fit, "x")
}

## End(Not run)
```

plot.prediction	<i>Plot prediction using ggplot2</i>
-----------------	--------------------------------------

Description

Generates a base ggplot2 using ggplot() and adds a geom for input x using [gg](#).

Usage

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

Arguments

x	a prediction object.
y	Ignored argument but required for S3 compatibility.
...	Arguments passed on to gg.prediction .

Details

Requires the ggplot2 package.

Value

an object of class gg

Examples

```
if (bru_safe_inla() && require(ggplot2, quietly = TRUE)) {  
  # Generate some data  
  
  input.df <- data.frame(x = cos(1:10))  
  input.df <- within(input.df, y <- 5 + 2 * cos(1:10) + rnorm(10, mean = 0, sd = 0.1))  
  
  # Fit a model with fixed effect 'x' and intercept 'Intercept'  
  
  fit <- bru(y ~ x, family = "gaussian", data = input.df)  
  
  # Predict posterior statistics of 'x'  
  
  xpost <- predict(fit, data = NULL, formula = ~x_latent)  
  
  # The statistics include mean, standard deviation, the 2.5% quantile, the median,  
  # the 97.5% quantile, minimum and maximum sample drawn from the posterior as well as  
  # the coefficient of variation and the variance.  
  
  xpost
```

```

# For a single variable like 'x' the default plotting method invoked by gg() will
# show these statistics in a fashion similar to a box plot:
ggplot() +
  gg(xpost)

# The predict function can also be used to simulataneously estimate posteriors
# of multiple variables:

xipost <- predict(fit,
  data = NULL,
  formula = ~ c(
    Intercept = Intercept_latent,
    x = x_latent
  )
)
xipost

# If we still want a plot in the previous style we have to set the bar parameter to TRUE

p1 <- ggplot() +
  gg(xipost, bar = TRUE)
p1

# Note that gg also understands the posterior estimates generated while running INLA

p2 <- ggplot() +
  gg(fit$summary.fixed, bar = TRUE)
multiplot(p1, p2)

# By default, if the prediction has more than one row, gg will plot the column 'mean' against
# the row index. This is for instance usefull for predicting and plotting function
# but not very meaningful given the above example:

ggplot() +
  gg(xipost)

# For ease of use we can also type

plot(xipost)

# This type of plot will show a ribbon around the mean, which viszualizes the upper and lower
# quantiles mentioned above (2.5 and 97.5%). Plotting the ribbon can be turned of using the
# \code{ribbon} parameter

ggplot() +
  gg(xipost, ribbon = FALSE)

# Much like the other geomes produced by gg we can adjust the plot using ggplot2 style
# commands, for instance

ggplot() +
  gg(xipost) +

```



```

    gg(xipost, mapping = aes(y = median), ribbon = FALSE, color = "red")
  }

```

plotsample

Create a plot sample.

Description

Creates a plot sample on a regular grid with a random start location.

Usage

```
plotsample(spdf, boundary, x.ppn = 0.25, y.ppn = 0.25, nx = 5, ny = 5)
```

Arguments

spdf	A <code>SpatialPointsDataFrame</code> defining the points that are to be sampled by the plot sample.
boundary	A <code>SpatialPolygonsDataFrame</code> defining the survey boundary within which the points occur.
x.ppn	The proportion of the x=axis that is to be included in the plots.
y.ppn	The proportion of the y=axis that is to be included in the plots.
nx	The number of plots in the x-dimension.
ny	The number of plots in the y-dimension.

Value

A list with three components:

plots: A `SpatialPolygonsDataFrame` object containing the plots that were sampled.

dets: A `SpatialPointsDataFrame` object containing the locations of the points within the plots.

counts: A dataframe containing the following columns

x: The x-coordinates of the centres of the plots within the boundary.

y: The y-coordinates of the centres of the plots within the boundary.

n: The numbers of points in each plot.

area: The areas of the plots within the boundary

Examples

```
# Some features require the raster package
if (require("raster", quietly = TRUE) &&
    require("ggplot2", quietly = TRUE)) {
  data(gorillas, package = "inlabru")
  plotpts <- plotsample(gorillas$nest, gorillas$boundary,
    x.ppn = 0.4, y.ppn = 0.4, nx = 5, ny = 5
  )
  ggplot() +
    gg(plotpts$plots) +
    gg(plotpts$dets, pch = "+", cex = 2) +
    gg(gorillas$boundary)
}
```

point2count

Convert a plot sample of points into one of counts.

Description

Converts a plot sample with locations of each point within each plot, into a plot sample with only the count within each plot.

Usage

```
point2count(plots, dets)
```

Arguments

plots	A SpatialPolygonsDataFrame object containing the plots that were sampled.
dets	A SpatialPointsDataFrame object containing the locations of the points within the plots.

Value

A SpatialPolygonsDataFrame with counts in each plot contained in slot @data\$.

Examples

```
# Some features require the raster package
if (require("raster", quietly = TRUE) &&
    require("ggplot2", quietly = TRUE)) {
  data(gorillas, package = "inlabru")
  plotpts <- plotsample(gorillas$nest, gorillas$boundary,
    x.ppn = 0.4, y.ppn = 0.4, nx = 5, ny = 5
  )
}
```

```

p1 <- ggplot() +
  gg(plotpts$plots) +
  gg(plotpts$dets) +
  gg(gorillas$boundary)
countdata <- point2count(plotpts$plots, plotpts$dets)
x <- coordinates(countdata)[, 1]
y <- coordinates(countdata)[, 2]
count <- countdata@data$n
p2 <- ggplot() +
  gg(gorillas$boundary) +
  gg(plotpts$plots) +
  geom_text(aes(label = count, x = x, y = y))
multiplot(p1, p2, cols = 2)
}

```

Poisson1_1D

1-Dimensional Homogeneous Poisson example.

Description

Point data and count data, together with intensity function and expected counts for a homogeneous 1-dimensional Poisson process example.

Usage

```
data(Poisson1_1D)
```

Format

The data contain the following R objects:

lambda1_1D: A function defining the intensity function of a nonhomogeneous Poisson process.

Note that this function is only defined on the interval (0,55).

E_nc1 The expected counts of the gridded data.

pts1 The locations of the observed points (a data frame with one column, named x).

countdata1 A data frame with three columns, containing the count data:

x The grid cell midpoint.

count The number of detections in the cell.

exposure The width of the cell.

Examples

```

if (require("ggplot2", quietly = TRUE)) {
  data(Poisson1_1D)
  ggplot(countdata1) +
    geom_point(data = countdata1, aes(x = x, y = count), col = "blue") +
    ylim(0, max(countdata1$count)) +
    geom_point(data = pts1, aes(x = x), y = 0.2, shape = "|", cex = 4) +
    geom_point(
      data = countdata1, aes(x = x), y = 0, shape = "+",
      col = "blue", cex = 4
    ) +
    xlab(expression(bold(s))) +
    ylab("count")
}

```

Poisson2_1D

1-Dimensional NonHomogeneous Poisson example.

Description

Point data and count data, together with intensity function and expected counts for a unimodal nonhomogeneous 1-dimensional Poisson process example.

Usage

```
data(Poisson2_1D)
```

Format

The data contain the following R objects:

lambda2_1D: A function defining the intensity function of a nonhomogeneous Poisson process.

Note that this function is only defined on the interval (0,55).

cov2_1D: A function that gives what we will call a 'habitat suitability' covariate in 1D space.

E_nc2 The expected counts of the gridded data.

pts2 The locations of the observed points (a data frame with one column, named x).

countdata2 A data frame with three columns, containing the count data:

x The grid cell midpoint.

count The number of detections in the cell.

exposure The width of the cell.

Examples

```

if (require("ggplot2", quietly = TRUE)) {
  data(Poisson2_1D)
  p1 <- ggplot(countdata2) +
    geom_point(data = countdata2, aes(x = x, y = count), col = "blue") +
    ylim(0, max(countdata2$count, E_nc2)) +
    geom_point(
      data = countdata2, aes(x = x), y = 0, shape = "+",
      col = "blue", cex = 4
    ) +
    geom_point(
      data = data.frame(x = countdata2$x, y = E_nc2), aes(x = x),
      y = E_nc2, shape = "_", cex = 5
    ) +
    xlab(expression(bold(s))) +
    ylab("count")
  ss <- seq(0, 55, length = 200)
  lambda <- lambda2_1D(ss)
  p2 <- ggplot() +
    geom_line(
      data = data.frame(x = ss, y = lambda),
      aes(x = x, y = y), col = "blue"
    ) +
    ylim(0, max(lambda)) +
    geom_point(data = pts2, aes(x = x), y = 0.2, shape = "|", cex = 4) +
    xlab(expression(bold(s))) +
    ylab(expression(lambda(bold(s))))
  multiplot(p1, p2, cols = 1)
}

```

Poisson3_1D

1-Dimensional NonHomogeneous Poisson example.

Description

Point data and count data, together with intensity function and expected counts for a multimodal nonhomogeneous 1-dimensional Poisson process example. Counts are given for two different gridded data interval widths.

Usage

```
data(Poisson3_1D)
```

Format

The data contain the following R objects:

lambda3_1D: A function defining the intensity function of a nonhomogeneous Poisson process.
Note that this function is only defined on the interval (0,55).

E_nc3a The expected counts of gridded data for the wider bins (10 bins).

E_nc3b The expected counts of gridded data for the wider bins (20 bins).

pts3 The locations of the observed points (a data frame with one column, named x).

countdata3a A data frame with three columns, containing the count data for the 10-interval case:

countdata3b A data frame with three columns, containing the count data for the 20-interval case:

x The grid cell midpoint.

count The number of detections in the cell.

exposure The width of the cell.

Examples

```
if (require("ggplot2", quietly = TRUE)) {
  data(Poisson3_1D)
  # first the plots for the 10-bin case:
  p1a <- ggplot(countdata3a) +
    geom_point(data = countdata3a, aes(x = x, y = count), col = "blue") +
    ylim(0, max(countdata3a$count, E_nc3a)) +
    geom_point(
      data = countdata3a, aes(x = x), y = 0, shape = "+",
      col = "blue", cex = 4
    ) +
    geom_point(
      data = data.frame(x = countdata3a$x, y = E_nc3a),
      aes(x = x), y = E_nc3a, shape = "_", cex = 5
    ) +
    xlab(expression(bold(s))) +
    ylab("count")
  ss <- seq(0, 55, length = 200)
  lambda <- lambda3_1D(ss)
  p2a <- ggplot() +
    geom_line(
      data = data.frame(x = ss, y = lambda), aes(x = x, y = y),
      col = "blue"
    ) +
    ylim(0, max(lambda)) +
    geom_point(data = pts3, aes(x = x), y = 0.2, shape = "|", cex = 4) +
    xlab(expression(bold(s))) +
    ylab(expression(lambda(bold(s))))
  multiplot(p1a, p2a, cols = 1)

  # Then the plots for the 20-bin case:
  p1a <- ggplot(countdata3b) +
    geom_point(data = countdata3b, aes(x = x, y = count), col = "blue") +
    ylim(0, max(countdata3b$count, E_nc3b)) +
    geom_point(
      data = countdata3b, aes(x = x), y = 0, shape = "+",
      col = "blue", cex = 4
    )
}
```

```

) +
geom_point(
  data = data.frame(x = countdata3b$x, y = E_nc3b),
  aes(x = x), y = E_nc3b, shape = "_", cex = 5
) +
xlab(expression(bold(s))) +
ylab("count")
ss <- seq(0, 55, length = 200)
lambda <- lambda3_1D(ss)
p2a <- ggplot() +
  geom_line(
    data = data.frame(x = ss, y = lambda), aes(x = x, y = y),
    col = "blue"
  ) +
  ylim(0, max(lambda)) +
  geom_point(data = pts3, aes(x = x), y = 0.2, shape = "|", cex = 4) +
  xlab(expression(bold(s))) +
  ylab(expression(lambda(bold(s))))
multiplot(p1a, p2a, cols = 1)
}

```

predict.bru

Prediction from fitted bru model

Description

Takes a fitted bru object produced by the function `bru()` and produces predictions given a new set of values for the model covariates or the original values used for the model fit. The predictions can be based on any R expression that is valid given these values/covariates and the joint posterior of the estimated random effects.

Usage

```

## S3 method for class 'bru'
predict(
  object,
  data = NULL,
  formula = NULL,
  n.samples = 100,
  seed = 0L,
  num.threads = NULL,
  include = NULL,
  exclude = NULL,
  drop = FALSE,
  ...
)

```

Arguments

object	An object obtained by calling <code>bru()</code> or <code>lgcp()</code> .
data	A <code>data.frame</code> or <code>SpatialPointsDataFrame</code> of covariates needed for the prediction.
formula	A formula where the right hand side defines an R expression to evaluate for each generated sample. If <code>NULL</code> , the latent and hyperparameter states are returned as named list elements. See Details for more information.
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 <code>inla.posterior.sample</code>
num.threads	Specification of desired number of threads for parallel computations. Default <code>NULL</code> , leaves it up to INLA. When <code>seed != 0</code> , overridden to "1:1"
include	Character vector of component labels that are needed by the predictor expression; Default: <code>NULL</code> (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: <code>NULL</code> (do not remove any components from the inclusion list)
drop	logical; If <code>keep=FALSE</code> , <code>data</code> is a <code>Spatial*DataFrame</code> , and the prediction summary has the same number of rows as <code>data</code> , then the output is a <code>Spatial*DataFrame</code> object. Default <code>FALSE</code> .
...	Additional arguments passed on to <code>inla.posterior.sample</code>

Details

Mean value predictions are accompanied by the standard errors, upper and lower 2.5% quantiles, the median, variance, coefficient of variation as well as the variance and minimum and maximum sample value drawn in course of estimating the statistics.

Internally, this method calls `generate.bru()` in order to draw samples from the model.

In addition to the component names (that give the effect of each component evaluated for the input data), the suffix `_latent` variable name can be used to directly access the latent state for a component, and the suffix function `_eval` can be used to evaluate a component at other input values than the expressions defined in the component definition itself, e.g. `field_eval(cbind(x,y))` for a component that was defined with `field(coordinates, ...)` (see also `component_eval()`).

For "iid" models with `mapper = bru_mapper_index(n)`, `rnorm()` is used to generate new realisations for indices greater than `n`.

Value

a `data.frame` or `Spatial*` object with predicted mean values and other summary statistics attached.

Examples

```
if (bru_safe_inla(multicore = FALSE) &&
    require("ggplot2", quietly = TRUE)) {
```



```

# Load the Gorilla data

data(gorillas, package = "inlabru")

# Plot the Gorilla nests, the mesh and the survey boundary

ggplot() +
  gg(gorillas$mesh) +
  gg(gorillas$nests) +
  gg(gorillas$boundary) +
  coord_fixed()

# Define SPDE prior

matern <- INLA::inla.spde2.pcmatern(gorillas$mesh,
  prior.sigma = c(0.1, 0.01),
  prior.range = c(0.01, 0.01)
)

# Define domain of the LGCP as well as the model components (spatial SPDE effect and Intercept)

cmp <- coordinates ~ mySmooth(main = coordinates, model = matern) + Intercept(1)

# Fit the model, with "eb" instead of full Bayes
fit <- lgcp(cmp, gorillas$nests,
  samplers = gorillas$boundary,
  domain = list(coordinates = gorillas$mesh),
  options = list(control.inla = list(int.strategy = "eb"))
)

# Once we obtain a fitted model the predict function can serve various purposes.
# The most basic one is to determine posterior statistics of a univariate
# random variable in the model, e.g. the intercept

icpt <- predict(fit, NULL, ~ c(Intercept = Intercept_latent))
plot(icpt)

# The formula argument can take any expression that is valid within the model, for
# instance a non-linear transformation of a random variable

exp.icpt <- predict(fit, NULL, ~ c(
  "Intercept" = Intercept_latent,
  "exp(Intercept)" = exp(Intercept_latent)
))
plot(exp.icpt, bar = TRUE)

# The intercept is special in the sense that it does not depend on other variables
# or covariates. However, this is not true for the smooth spatial effects 'mySmooth'.
# In order to predict 'mySmooth' we have to define where (in space) to predict. For
# this purpose, the second argument of the predict function can take \code{data.frame}
# objects as well as Spatial objects. For instance, we might want to predict
# 'mySmooth' at the locations of the mesh vertices. Using

```

```

vrt <- vertices(gorillas$mesh)

# we obtain these vertices as a SpatialPointsDataFrame

ggplot() +
  gg(gorillas$mesh) +
  gg(vrt, color = "red")

# Predicting 'mySmooth' at these locations works as follows

mySmooth <- predict(fit, vrt, ~mySmooth)

# Note that just like the input also the output will be a SpatialPointsDataFrame
# and that the predicted statistics are simply added as columns

class(mySmooth)
head(vrt)
head(mySmooth)

# Plotting the mean, for instance, at the mesh node is straight forward

ggplot() +
  gg(gorillas$mesh) +
  gg(mySmooth, aes(color = mean), size = 3)

# However, we are often interested in a spatial field and thus a linear interpolation,
# which can be achieved by using the gg mechanism for meshes

ggplot() +
  gg(gorillas$mesh, color = mySmooth$mean)

# Alternatively, we can predict the spatial field at a grid of locations, e.g. a
# SpatialPixels object covering the mesh

pxl <- pixels(gorillas$mesh)
mySmooth2 <- predict(fit, pxl, ~mySmooth)

# This will give us a SpatialPixelDataFrame with the columns we are looking for

head(mySmooth2)
ggplot() +
  gg(mySmooth2)
}

```

Description

Takes two Matrices and computes the row-wise Kronecker product. Optionally applies row-wise weights and/or applies an additional 0/1 row-wise Kronecker matrix product.

Usage

```
row_kron(M1, M2, repl = NULL, n.repl = NULL, weights = NULL)
```

Arguments

M1	A matrix that can be transformed into a sparse Matrix.
M2	A matrix that can be transformed into a sparse Matrix.
repl	An optional index vector. For each entry, specifies which replicate the row belongs to, in the sense used in <code>INLA::inla.spde.make.A</code>
n.repl	The maximum replicate index, in the sense used in <code>INLA::inla.spde.make.A()</code> .
weights	Optional scaling weights to be applied row-wise to the resulting matrix.

Value

A `Matrix::sparseMatrix` object.

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

sample.lgcp

Sample from an inhomogeneous Poisson process

Description

This function provides point samples from one- and two-dimensional inhomogeneous Poisson processes. The log intensity has to be provided via its values at the nodes of an `inla.mesh.1d` or `inla.mesh` object. In between mesh nodes the log intensity is assumed to be linear.

Usage

```
sample.lgcp(
  mesh,
  loglambda,
  strategy = NULL,
  R = NULL,
  samplers = NULL,
  ignore.CRS = FALSE
)
```

Arguments

mesh	An <code>INLA::inla.mesh</code> object
loglambda	vector or matrix; A vector of log intensities at the mesh vertices (for higher order basis functions, e.g. for <code>inla.mesh.1d</code> meshes, loglambda should be given as <code>mesh\$m</code> basis function weights rather than the values at the <code>mesh\$n</code> vertices) A single scalar is expanded to a vector of the appropriate length. If a matrix is supplied, one process sample for each column is produced.
strategy	Only relevant for 2D meshes. One of 'triangulated', 'rectangle', 'sliced-spherical', 'spherical'. The 'rectangle' method is only valid for CRS-less flat 2D meshes. If NULL or 'auto', the the likely fastest method is chosen; 'rectangle' for flat 2D meshes with no CRS, 'sliced-spherical' for CRS 'longlat' meshes, and 'triangulated' for all other meshes.
R	Numerical value only applicable to spherical and geographical meshes. It is interpreted as R is the equivalent Earth radius, in km, used to scale the lambda intensity. For CRS enabled meshes, the default is 6371. For CRS-less spherical meshes, the default is 1.
samplers	A <code>SpatialPolygonsDataFrame</code> or <code>inla.mesh</code> object. Simulated points that fall outside these polygons are discarded.
ignore.CRS	logical; if TRUE, ignore any CRS information in the mesh. Default FALSE. This affects R and the permitted values for strategy.

Details

For 2D processes on a sphere the R parameter can be used to adjust to sphere's radius implied by the mesh. If the intensity is very high the standard strategy "spherical" can cause memory issues. Using the "sliced-spherical" strategy can help in this case.

- For crs-less meshes on R2: Lambda is interpreted in the raw coordinate system. Output has an NA CRS.
- For crs-less meshes on S2: Lambda with raw units, after scaling the mesh to radius R, if specified. Output is given on the same domain as the mesh, with an NA CRS.
- For crs meshes on R2: Lambda is interpreted as per km^2 , after scaling the globe to the Earth radius 6371 km, or R, if specified. Output given in the same CRS as the mesh.
- For crs meshes on S2: Lambda is interpreted as per km^2 , after scaling the globe to the Earth radius 6371 km, or R, if specified. Output given in the same CRS as the mesh.

Value

A `data.frame` (1D case), `SpatialPoints` (2D flat and 3D spherical surface cases) `SpatialPoints-DataFrame` (2D/3D surface cases with multiple samples). For multiple samples, the `data.frame` output has a column 'sample' giving the index for each sample. object of point locations.

Author(s)

Daniel Simpson <dp.simpson@gmail.com> (base rectangle and spherical algorithms), Fabian E. Bachl <bachlfab@gmail.com> (inclusion in `inlabru`, sliced spherical sampling), Finn Lindgren <finn.lindgren@gmail.com> (extended CRS support, triangulated sampling)

Examples

```
# The INLA package is required
if (bru_safe_inla(quietly = TRUE)) {
  vertices <- seq(0, 3, by = 0.1)
  mesh <- INLA::inla.mesh.1d(vertices)
  loglambda <- 5 - 0.5 * vertices
  pts <- sample.lgcp(mesh, loglambda)
  pts$y <- 0
  plot(vertices, exp(loglambda), type = "l", ylim = c(0, 150))
  points(pts, pch = "|")
}
```

```
# The INLA package and PROJ6 are required
if (bru_safe_inla(quietly = TRUE) &&
    fm_has_PROJ6() &&
    require(ggplot2, quietly = TRUE)) {
  data("gorillas", package = "inlabru")
  pts <- sample.lgcp(gorillas$mesh,
    loglambda = 1.5,
    samplers = gorillas$boundary
  )
  ggplot() +
    gg(gorillas$mesh) +
    gg(pts)
}
```

 seals

Seal pups

Description

This is a single transect of an aerial photo seal pup survey in the Greenland Sea

Usage

```
data(seals)
```

Format

The data contain these objects:

points: A `SpatialPointsDataFrame` Center locations of the photos

mesh: An `inla.mesh` enclosing the plane's transect

ice.data: An `SpatialPointsDataFrame` with MODIS ice concentration estimates

ice.cv: An `covdata` object with interpolated ice coverage data

Source

Martin Jullum <Martin.Jullum@nr.no>

References

Oigard, T. A. (2013) From pup production to quotas: current status of harp seals in the Greenland Sea. ICES Journal of Marine Science, doi.10.1093/icesjms/fst155.

Oigard, T. A. (2014) Current status of hooded seals in the Greenland Sea. Victims of climate change and predation?, Biological Conservation , 2014, 172, 29 - 36.

Examples

```
if (require(ggplot2, quietly = TRUE)) {
  data(seals, package = "inlabru")
  ggplot() +
    gg(seals$mesh) +
    gg(seals$points)
}
```

 shrimp

Blue and red shrimp in the Western Mediterranean Sea

Description

Blue and red shrimp in the Western Mediterranean Sea.

Usage

```
data(shrimp)
```

Format

A list of objects:

haul: A SpatialPointsDataFrame object containing haul locations

mesh: An inla.mesh object containing a Delaunay triangulation mesh (a type of discretization of continuous space) covering the haul locations.

catch Catch in Kg.

landing Landing in Kg.

depth Mean depth of the fishery haul.

Source

Pennino, Maria Grazia. Personal communication.

References

Pennino, M. G., Paradinas, I., Munoz, F., Illian, J., Quilez-Lopez, A., Bellido, J.M., Conesa, D. Accounting for preferential sampling in species distribution models. *Ecology and Evolution*, In Press.

Examples

```
if (require(ggplot2, quietly = TRUE)) {
  data(shrimp, package = "inlabru")
  ggplot() +
    gg(shrimp$mesh) +
    gg(shrimp$hauls) +
    coord_equal()
}
```

sline

Convert data frame to SpatialLinesDataFrame

Description

A line in 2D space is defined by a start and an end point, each associated with 2D coordinates. This function takes a `/codedata.frame` as input and assumes that each row defines a line in space. In order to do so, the data frame must have at least four columns and the `start.cols` and `end.cols` parameters must be used to point out the names of the columns that define the start and end coordinates of the line. The data is then converted to a `SpatialLinesDataFrame` DF. If a coordinate reference system `crs` is provided it is attached to DF. If also `to.crs` is provided, the coordinate system of DF is transformed accordingly. Additional columns of the input data, e.g. covariates, are retained and attached to DF.

Usage

```
sline(data, start.cols, end.cols, crs = CRS(as.character(NA)), to.crs = NULL)
```

Arguments

<code>data</code>	A data.frame
<code>start.cols</code>	Character array pointing out the columns of data that hold the start points of the lines
<code>end.cols</code>	Character array pointing out the columns of data that hold the end points of the lines
<code>crs</code>	Coordinate reference system of the original data
<code>to.crs</code>	Coordinate reference system for the <code>SpatialLines</code> output.

Value

SpatialLinesDataFrame

Examples

```
# Create a data frame defining three lines
lms <- data.frame(
  xs = c(1, 2, 3), ys = c(1, 1, 1), # start points
  xe = c(2, 3, 4), ye = c(2, 2, 2)
) # end points

# Conversion to SpatialLinesDataFrame without CRS
spl <- sline(lms,
  start.cols = c("xs", "ys"),
  end.cols = c("xe", "ye")
)

if (require(ggplot2, quietly = TRUE)) {
  # Plot the lines
  ggplot() +
    gg(spl)
}
```

spatial.to.ppp

Convert SpatialPoints and boundary polygon to spatstat ppp object

Description

Spatstat point pattern objects consist of points and an observation windows. This function uses a SpatialPoints object and a SpatialPolygon object to generate the points and the window. Lastly, the ppp() function is called to create the ppp object.

Usage

```
spatial.to.ppp(points, samplers)
```

Arguments

points A SpatialPoints[DataFrame] object describing the point pattern.
samplers A SpatialPolygons[DataFrame] object describing the observation window.

Value

A spatstat spatstat ppp object

Examples

```

if (require("spatstat.geom")) {
  # Load Gorilla data

  data("gorillas", package = "inlabru")

  # Use nest locations and survey boundary to create a spatstat ppp object

  gp <- spatial.to.ppp(gorillas$nest, gorillas$boundary)
  class(gp)

  # Plot it

  plot(gp)
}

```

spde.posterior	<i>Posteriors of SPDE hyper parameters and Matern correlation or covariance function.</i>
----------------	---

Description

Calculate posterior distribution of the range, $\log(\text{range})$, variance, or $\log(\text{variance})$ parameter of a model's SPDE component. Can also plot Matern correlation or covariance function. `inla.spde.result`.

Usage

```
spde.posterior(result, name, what = "range")
```

Arguments

result	An object inheriting from <code>inla</code> .
name	Character stating the name of the SPDE effect, see <code>names(result\$summary.random)</code> .
what	One of "range", "log.range", "variance", "log.variance", "matern.correlation" or "matern.covariance".

Value

A prediction object.

Author(s)

Finn Lindgren <Finn.Lindgren@ed.ac.uk>

Examples

```

if (bru_safe_inla() && require(ggplot2, quietly = TRUE)) {

  # Load 1D Poisson process data

  data(Poisson2_1D, package = "inlabru")

  # Take a look at the point (and frequency) data

  ggplot(pts2) +
    geom_histogram(aes(x = x), binwidth = 55 / 20, boundary = 0, fill = NA, color = "black") +
    geom_point(aes(x, y = 0, pch = "|", cex = 4) +
      coord_fixed(ratio = 1)

  # Fit an LGCP model with and SPDE component

  x <- seq(0, 55, length = 20)
  mesh1D <- INLA::inla.mesh.1d(x, boundary = "free")
  mdl <- x ~ spde1D(x, model = INLA::inla.spde2.matern(mesh1D)) + Intercept
  fit <- lgcp(mdl, data = pts2, domain = list(x = mesh1D))

  # Calculate and plot the posterior range

  range <- spde.posterior(fit, "spde1D", "range")
  plot(range)

  # Calculate and plot the posterior log range

  lrange <- spde.posterior(fit, "spde1D", "log.range")
  plot(lrange)

  # Calculate and plot the posterior variance

  variance <- spde.posterior(fit, "spde1D", "variance")
  plot(variance)

  # Calculate and plot the posterior log variance

  lvariance <- spde.posterior(fit, "spde1D", "log.variance")
  plot(lvariance)

  # Calculate and plot the posterior Matern correlation

  matcor <- spde.posterior(fit, "spde1D", "matern.correlation")
  plot(matcor)

  # Calculate and plot the posterior Matern covariance

  matcov <- spde.posterior(fit, "spde1D", "matern.covariance")
  plot(matcov)

```

```
}

```

spoly	<i>Convert a data.frame of boundary points into a SpatialPolygons-DataFrame</i>
-------	---

Description

A polygon can be described as a sequence of points defining the polygon's boundary. When given such a sequence (anti clockwise!) this function creates a `SpatialPolygonsDataFrame` holding the polygon described. By default, the first two columns of data are assumed to define the x and y coordinates of the points. This behavior can be changed using the `cols` parameter, which points out the names of the columns holding the coordinates. The coordinate reference system of the resulting spatial polygon can be set via the `crs` parameter. Posterior conversion to a different CRS is supported using the `to.crs` parameter.

Usage

```
spoly(
  data,
  cols = colnames(data)[1:2],
  crs = CRS(NA_character_),
  to.crs = NULL
)
```

Arguments

<code>data</code>	A data.frame of points describing the boundary of the polygon
<code>cols</code>	Column names of the x and y coordinates within the data
<code>crs</code>	Coordinate reference system of the points
<code>to.crs</code>	Coordinate reference system for the <code>SpatialLines</code> output.

Value

`SpatialPolygonsDataFrame`

Examples

```
# Create data frame of boundary points (anti clockwise!)
pts <- data.frame(
  x = c(1, 2, 1.7, 1.3),
  y = c(1, 1, 2, 2)
)

# Convert to SpatialPolygonsDataFrame
pol <- spoly(pts)
```

```

if (require(ggplot2, quietly = TRUE)) {
  # Plot it!
  ggplot() +
    gg(pol)
}

```

stransform

Coordinate transformation for spatial objects

Description

This is a wrapper for the [spTransform](#) function provided by the `sp` package. Given a spatial object (or a list thereof) it will transform the coordinate system according to the parameter `crs`. In addition to the usual spatial objects this function is also capable of transforming `INLA::inla.mesh` objects that are equipped with a coordinate system. #'

Usage

```
stransform(splist, crs)
```

Arguments

<code>splist</code>	list of Spatial* objects
<code>crs</code>	Coordinate reference system to change to

Value

List of Spatial* objects

Examples

```

# Load Gorilla data
data("gorillas", package = "inlabru")

# Take the mesh and transform it to latitude/longitude
tmesh <- stransform(gorillas$mesh, crs = CRS("+proj=longlat"))

# Compare original and transformed mesh

if (require(ggplot2, quietly = TRUE)) {
  multiplot(
    ggplot() +
      gg(gorillas$mesh) +
      ggtitle("Original mesh"),
    ggplot() +

```

```

    gg(tmesh) +
      ggtitle("Transformed mesh")
  )
}

```

summary.bru

Summary for an inlabru fit

Description

Takes a fitted bru object produced by `bru()` or `lgcp()` and creates various summaries from it.

Usage

```

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

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

```

Arguments

object	An object obtained from a <code>bru()</code> or <code>lgcp()</code> call
...	ignored arguments
x	An <code>summary_bru2</code> object

Examples

```

if (bru_safe_inla(multicore = FALSE)) {

  # Simulate some covariates x and observations y
  input.df <- data.frame(x = cos(1:10))
  input.df <- within(input.df, y <- 5 + 2 * x + rnorm(10, mean = 0, sd = 0.1))

  # Fit a Gaussian likelihood model
  fit <- bru(y ~ x + Intercept, family = "gaussian", data = input.df)

  # Obtain summary
  fit$summary.fixed
}

if (bru_safe_inla(multicore = FALSE)) {

  # Alternatively, we can use the like() function to construct the likelihood:

```

```

lik <- like(family = "gaussian", formula = y ~ x + Intercept, data = input.df)
fit <- bru(~ x + Intercept(1), lik)
fit$summary.fixed
}

# An important addition to the INLA methodology is bru's ability to use
# non-linear predictors. Such a predictor can be formulated via like()'s
# \code{formula} parameter. The z(1) notation is needed to ensure that
# the z component should be interpreted as single latent variable and not
# a covariate:

if (bru_safe_inla(multicore = FALSE)) {
  z <- 2
  input.df <- within(input.df, y <- 5 + exp(z) * x + rnorm(10, mean = 0, sd = 0.1))
  lik <- like(
    family = "gaussian", data = input.df,
    formula = y ~ exp(z) * x + Intercept
  )
  fit <- bru(~ z(1) + Intercept(1), lik)

  # Check the result (z posterior should be around 2)
  fit$summary.fixed
}

```

summary.bru_info

Methods for bru_info objects

Description

Methods for bru_info objects

Summary for bru_info objects

Usage

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

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

```
bru_info(...)
```

```
## S3 method for class 'character'
bru_info(method, ..., inlabru_version = NULL, INLA_version = NULL)
```

```
## S3 method for class 'bru'
bru_info(object, ...)
```

Arguments

object	Object to operate on
...	Arguments passed on to other methods
x	A summary_bru_info object to be printed
method	character; The type of estimation method used
inlabru_version	character; inlabru package version. Default: NULL, for automatically detecting the version
INLA_version	character; INLA package version. Default: NULL, for automatically detecting the version

summary.bru_options *Print inlabru options*

Description

Print inlabru options

Usage

```
## S3 method for class 'bru_options'
summary(
  object,
  legend = TRUE,
  include_global = TRUE,
  include_default = TRUE,
  ...
)

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

Arguments

object	A bru_options object to be summarised
legend	logical; If TRUE, include explanatory text, Default: TRUE
include_global	logical; If TRUE, include global override options
include_default	logical; If TRUE, include default options
...	Further parameters, currently ignored
x	A summary_bru_options object to be printed

Examples

```

if (interactive()) {
  options <- bru_options(verbose = TRUE)

  # Don't print options only set in default:
  print(options, include_default = FALSE)

  # Only include options set in the object:
  print(options, include_default = FALSE, include_global = FALSE)
}

```

summary_bru

Deprecated functions in inlabru

Description

These functions still attempt to do their job, but will be removed in a future version.

Usage

```

summary_bru(object, ...)

iinla.getOption(name = NULL)

iinla.setOption(...)

init.tutorial()

```

Arguments

object	An object obtained from a <code>bru()</code> or <code>lgcp()</code> call
...	arguments passed on to other methods or ignored
name	character; an option name

Functions

- `summary_bru`: Old summary for an inlabru fit. Takes a fitted bru object produced by `bru()` or `lgcp()` and creates various summaries from it.
- `iinla.getOption`: Use `bru_option_get` instead.
- `iinla.setOption`: Use `bru_option_set` instead.
- `init.tutorial`: Global setting for tutorial sessions. Use `bru_options_set()` to set specific options instead. In versions $\leq 2.1.15$, this function set the INLA integration strategy to "eb" to speed up calculations. This is normally not needed since version 2.2.0, since the only the final iteration will use other than "eb".

Author(s)

Fabian E. Bachl <bachlfab@gmail.com>

Examples

```
## Not run:  
# Note: Only run this if you want to change the inlabru options for this session  
  
# Determine current bru defaults:  
bo <- bru_options_get()  
  
init.tutorial()  
  
# Check if it worked:  
bru_options_get("control.inla")  
  
## End(Not run)
```

toygroups

Simulated 1D animal group locations and group sizes

Description

This data set serves to teach the concept of modelling species that gather in groups and where the grouping behaviour depends on space.

Usage

```
data(toygroups)
```

Format

The data are a list that contains these elements:

`groups`: A `data.frame` of group locations `x` and size `size`

`df.size`: IGNORE THIS

`df.intensity`: A `data.frame` with Poisson process intensity `d.lambda` at locations `x`

`df.rate`: A `data.frame` the locations `x` and associated rate which parameterized the exponential distribution from which the group sizes were drawn.

Examples

```

if (require(ggplot2, quietly = TRUE)) {
  # Load the data

  data("toygroups", package = "inlabru")

  # The data set is a simulation of animal groups residing in a 1D space. Their
  # locations in x-space are sampled from a Cox process with intensity

  ggplot(toygroups$df.intensity) +
    geom_line(aes(x = x, y = g.lambda))

  # Adding the simulated group locations to this plot we obtain

  ggplot(toygroups$df.intensity) +
    geom_line(aes(x = x, y = g.lambda)) +
    geom_point(data = toygroups$groups, aes(x, y = 0), pch = "|")

  # Each group has a size mark attached to it.
  # These group sizes are sampled from an exponential distribution
  # for which the rate parameter depends on the x-coordinate

  ggplot(toygroups$groups) +
    geom_point(aes(x = x, y = size))

  ggplot(toygroups$df.rate) +
    geom_line(aes(x, rate))
}

```

vertices

Vertices

Description

This is a generic function. The outcome depends on the object provided

Usage

```
vertices(object)
```

```
## S4 method for signature 'inla.mesh'
vertices(object)
```

Arguments

object An object for which to call the particular vertices method.

Value

The form of the value returned by vertices() depends on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

vertices.inla.mesh *Extract vertex locations from an inla.mesh*

Description

Converts the vertices of an inla.mesh object into a SpatialPointsDataFrame.

Usage

```
vertices.inla.mesh(object)
```

Arguments

object An inla.mesh object.

Value

A SpatialPointsDataFrame of mesh vertex locations. The vrt column indicates the internal vertex id.

Author(s)

Fabian E. Bachl <bachlfab@gmail.com>

Examples

```
if (require(ggplot2, quietly = TRUE)) {
  data("mrsea", package = "inlabru")
  vrt <- vertices(mrsea$mesh)
  ggplot() +
    gg(mrsea$mesh) +
    gg(vrt, color = "red")
}
```

Index

- * **component constructors**
 - component, 26
 - component_list, 31
- * **geomes for Raster data**
 - gg, 49
 - gg.RasterLayer, 59
 - gm, 75
- * **geomes for inla and inlabru predictions**
 - gg, 49
 - gg.data.frame, 50
 - gg.matrix, 56
 - gg.prediction, 57
 - gm, 75
- * **geomes for meshes**
 - gg, 49
 - gg.inla.mesh, 52
 - gg.inla.mesh.1d, 54
 - gm, 75
- * **geomes for spatial data**
 - gg, 49
 - gg.SpatialGridDataFrame, 60
 - gg.SpatialLines, 62
 - gg.SpatialPixels, 63
 - gg.SpatialPixelsDataFrame, 64
 - gg.SpatialPoints, 66
 - gg.SpatialPolygons, 68
 - gm, 75
- * **inlabru RGL tools**
 - globe, 70
 - glplot, 71
 - glplot.inla.mesh, 72
 - glplot.SpatialLines, 73
 - glplot.SpatialPoints, 74
- * **sample generators**
 - generate, 46
- [.bru_like_list (like), 86
- [.bru_mapper_collect
 - (bru_mapper_methods), 17
- [.bru_mapper_multi
 - (bru_mapper_methods), 17
- [.component_list (component_list), 31
- as.bru_options (bru_options), 22
- as.bru_options(), 8
- base:::makeMessage(), 13
- bincount, 4
- bru, 6
- bru(), 5, 22, 23, 26, 28, 29, 34, 46, 47, 80, 85–88, 94, 103, 104, 117, 120
- bru_call_options, 8
- bru_component, 9
- bru_component (component), 26
- bru_component_list (component_list), 31
- bru_compute_linearisation, 8
- bru_fill_missing, 10
- bru_forward_transformation, 11
- bru_info (summary.bru_info), 118
- bru_inverse_transformation
 - (bru_forward_transformation), 11
- bru_log (bru_log_reset), 12
- bru_log_active (bru_log_reset), 12
- bru_log_active(), 13
- bru_log_get (bru_log_reset), 12
- bru_log_message (bru_log_reset), 12
- bru_log_message(), 23
- bru_log_reset, 12
- bru_make_stack, 14
- bru_mapper, 15
- bru_mapper(), 21
- bru_mapper_collect (bru_mapper), 15
- bru_mapper_factor (bru_mapper), 15
- bru_mapper_index (bru_mapper), 15
- bru_mapper_linear (bru_mapper), 15
- bru_mapper_matrix (bru_mapper), 15
- bru_mapper_methods, 17, 17
- bru_mapper_multi (bru_mapper), 15
- bru_mapper_offset (bru_mapper), 15

- bru_options, [6](#), [22](#), [22](#), [23](#), [88](#), [119](#)
- bru_options(), [6](#), [24](#), [88](#)
- bru_options_check (bru_options), [22](#)
- bru_options_default (bru_options), [22](#)
- bru_options_default(), [24](#)
- bru_options_get (bru_options), [22](#)
- bru_options_get(), [24](#)
- bru_options_reset (bru_options), [22](#)
- bru_options_set (bru_options), [22](#)
- bru_options_set(), [13](#), [85](#), [120](#)
- bru_rerun (bru), [6](#)
- bru_safe_inla, [25](#)
- bru_standardise_names, [26](#)

- c.component_list (component_list), [31](#)
- class, [49](#)
- component, [26](#), [32](#)
- component(), [6](#), [10](#), [31](#), [80](#)
- component.character(), [26](#)
- component_eval, [30](#)
- component_eval(), [47](#), [87](#), [104](#)
- component_list, [30](#), [31](#)
- component_list.formula(), [26](#)
- countdata1 (Poisson1_1D), [99](#)
- countdata2 (Poisson2_1D), [100](#)
- countdata3a (Poisson3_1D), [101](#)
- countdata3b (Poisson3_1D), [101](#)
- cov2_1D (Poisson2_1D), [100](#)
- cprod, [33](#)
- CRS, [53](#), [65](#), [66](#)

- deltaIC, [34](#)
- devel.cvmeasure, [35](#)

- E_nc1 (Poisson1_1D), [99](#)
- E_nc2 (Poisson2_1D), [100](#)
- E_nc3a (Poisson3_1D), [101](#)
- E_nc3b (Poisson3_1D), [101](#)
- evaluate_A, [38](#)
- evaluate_index, [38](#)
- expand_labels, [39](#)

- fm_CRS, [39](#)
- fm_CRS(), [43](#)
- fm_crs_get_ellipsoid_radius (fm_wkt_is_geocent), [45](#)
- fm_crs_get_lengthunit (fm_wkt_is_geocent), [45](#)
- fm_crs_get_wkt (fm_wkt_is_geocent), [45](#)
- fm_crs_is_geocent (fm_wkt_is_geocent), [45](#)
- fm_crs_set_ellipsoid_radius (fm_wkt_is_geocent), [45](#)
- fm_crs_set_lengthunit (fm_wkt_is_geocent), [45](#)
- fm_crs_wkt, [41](#)
- fm_crs_wkt (fm_wkt_is_geocent), [45](#)
- fm_fallback_PROJ6 (fm_has_PROJ6), [41](#)
- fm_has_PROJ6, [41](#)
- fm_identical_CRS(), [41](#)
- fm_not_for_PROJ4 (fm_has_PROJ6), [41](#)
- fm_not_for_PROJ6 (fm_has_PROJ6), [41](#)
- fm_requires_PROJ6 (fm_has_PROJ6), [41](#)
- fm_sp_get_crs, [43](#)
- fm_sp_get_crs(), [41](#), [46](#)
- fm_spTransform, [42](#)
- fm_wkt_as_wkt_tree, [44](#)
- fm_wkt_get_ellipsoid_radius (fm_wkt_is_geocent), [45](#)
- fm_wkt_get_lengthunit (fm_wkt_is_geocent), [45](#)
- fm_wkt_is_geocent, [45](#)
- fm_wkt_predef (fm_CRS), [39](#)
- fm_wkt_set_ellipsoid_radius (fm_wkt_is_geocent), [45](#)
- fm_wkt_set_lengthunit (fm_wkt_is_geocent), [45](#)
- fm_wkt_tree_as_wkt (fm_wkt_as_wkt_tree), [44](#)
- fm_wkt_tree_get_item (fm_wkt_as_wkt_tree), [44](#)
- fm_wkt_tree_set_item (fm_wkt_as_wkt_tree), [44](#)
- fm_wkt_unit_params (fm_wkt_is_geocent), [45](#)

- generate, [46](#)
- generate.bru(), [104](#)
- gg, [49](#), [51](#), [53](#), [55–57](#), [60–62](#), [64](#), [65](#), [67](#), [68](#), [75](#), [76](#), [95](#)
- gg(), [75](#), [80](#)
- gg.data.frame, [50](#), [50](#), [56](#), [57](#), [76](#)
- gg.inla.mesh, [50](#), [52](#), [55](#), [76](#)
- gg.inla.mesh.1d, [50](#), [53](#), [54](#), [76](#)
- gg.matrix, [50](#), [51](#), [56](#), [57](#), [76](#)
- gg.prediction, [50](#), [51](#), [56](#), [57](#), [76](#), [95](#)
- gg.prediction(), [50](#)
- gg.RasterLayer, [50](#), [59](#), [76](#)

- gg.SpatialGridDataFrame, [50](#), [60](#), [62](#), [64](#), [65](#), [67](#), [68](#), [76](#)
- gg.SpatialLines, [50](#), [61](#), [62](#), [64](#), [65](#), [67](#), [68](#), [76](#)
- gg.SpatialPixels, [50](#), [61](#), [62](#), [63](#), [65](#), [67](#), [68](#), [76](#)
- gg.SpatialPixelsDataFrame, [50](#), [61](#), [62](#), [64](#), [64](#), [67](#), [68](#), [76](#)
- gg.SpatialPixelsDataFrame(), [52](#), [53](#), [60](#)
- gg.SpatialPoints, [50](#), [61](#), [62](#), [64](#), [65](#), [66](#), [68](#), [76](#)
- gg.SpatialPolygons, [50](#), [61](#), [62](#), [64](#), [65](#), [67](#), [68](#), [76](#)
- globe, [70](#), [71–74](#)
- glplot, [70](#), [71](#), [72–74](#)
- glplot.inla.mesh, [70](#), [71](#), [72](#), [73](#), [74](#)
- glplot.SpatialLines, [70–72](#), [73](#), [74](#)
- glplot.SpatialPoints, [70–73](#), [74](#)
- gm, [50](#), [51](#), [53](#), [55–57](#), [60–62](#), [64](#), [65](#), [67](#), [68](#), [75](#)
- gmap, [75](#), [76](#)
- gorillas, [77](#), [80](#)
- ibm_amatrix (bru_mapper), [15](#)
- ibm_amatrix.bru_mapper_collect (bru_mapper_methods), [17](#)
- ibm_amatrix.bru_mapper_factor (bru_mapper_methods), [17](#)
- ibm_amatrix.bru_mapper_index (bru_mapper_methods), [17](#)
- ibm_amatrix.bru_mapper_inla_mesh_1d (bru_mapper_methods), [17](#)
- ibm_amatrix.bru_mapper_inla_mesh_2d (bru_mapper_methods), [17](#)
- ibm_amatrix.bru_mapper_linear (bru_mapper_methods), [17](#)
- ibm_amatrix.bru_mapper_matrix (bru_mapper_methods), [17](#)
- ibm_amatrix.bru_mapper_multi (bru_mapper_methods), [17](#)
- ibm_amatrix.bru_mapper_offset (bru_mapper_methods), [17](#)
- ibm_amatrix.default (bru_mapper_methods), [17](#)
- ibm_inla_subset (bru_mapper), [15](#)
- ibm_n (bru_mapper), [15](#)
- ibm_n.bru_mapper_collect (bru_mapper_methods), [17](#)
- ibm_n.bru_mapper_factor (bru_mapper_methods), [17](#)
- ibm_n.bru_mapper_inla_mesh_1d (bru_mapper_methods), [17](#)
- ibm_n.bru_mapper_inla_mesh_2d (bru_mapper_methods), [17](#)
- ibm_n.bru_mapper_linear (bru_mapper_methods), [17](#)
- ibm_n.bru_mapper_matrix (bru_mapper_methods), [17](#)
- ibm_n.bru_mapper_multi (bru_mapper_methods), [17](#)
- ibm_n.bru_mapper_offset (bru_mapper_methods), [17](#)
- ibm_n.default (bru_mapper_methods), [17](#)
- ibm_values (bru_mapper), [15](#)
- ibm_values.bru_mapper_collect (bru_mapper_methods), [17](#)
- ibm_values.bru_mapper_factor (bru_mapper_methods), [17](#)
- ibm_values.bru_mapper_inla_mesh_1d (bru_mapper_methods), [17](#)
- ibm_values.bru_mapper_inla_mesh_2d (bru_mapper_methods), [17](#)
- ibm_values.bru_mapper_linear (bru_mapper_methods), [17](#)
- ibm_values.bru_mapper_matrix (bru_mapper_methods), [17](#)
- ibm_values.bru_mapper_multi (bru_mapper_methods), [17](#)
- ibm_values.bru_mapper_offset (bru_mapper_methods), [17](#)
- ibm_values.default (bru_mapper_methods), [17](#)
- iinla.getOption (summary_bru), [120](#)
- iinla.setOption (summary_bru), [120](#)
- init.tutorial (summary_bru), [120](#)
- inla.stack.mjoin, [79](#)
- inlabru, [80](#)
- inlabru-deprecated (summary_bru), [120](#)
- input_eval.bru_input(), [16](#)

int, 80
ipoints, 33, 81

lambda1_1D (Poisson1_1D), 99
lambda2_1D (Poisson2_1D), 100
lambda3_1D (Poisson3_1D), 101
lgcp, 6, 84
lgcp(), 5, 23, 34, 80, 87, 104, 117, 120
like, 6, 86
like(), 6, 80, 85
like_list (like), 86

mexdolphins, 80, 89
mrsea, 91
multiplot, 92

names.bru_mapper_collect
 (bru_mapper_methods), 17
names.bru_mapper_collect(), 21
names.bru_mapper_multi
 (bru_mapper_methods), 17
names.bru_mapper_multi(), 21
names<- .bru_mapper_collect
 (bru_mapper_methods), 17
names<- .bru_mapper_multi
 (bru_mapper_methods), 17

pixels, 93
plot.bru, 94
plot.prediction, 95
plotsample, 97
point2count, 98
Poisson1_1D, 80, 99
Poisson2_1D, 80, 100
Poisson3_1D, 101
predict(), 5
predict.bru, 29, 48, 103
predict.bru(), 5, 57, 80
print.summary_bru (summary.bru), 117
print.summary_bru_info
 (summary.bru_info), 118
print.summary_bru_options
 (summary.bru_options), 119
pts1 (Poisson1_1D), 99
pts2 (Poisson2_1D), 100
pts3 (Poisson3_1D), 101

row_kron, 106
sample.lgcp, 107

seals, 80, 109
shrimp, 110
sline, 111
sp::CRS(), 41
spatial.to.ppp, 112
SpatialPixels, 64
spde.posterior, 113
spoly, 115
spTransform, 116
strtransform, 116
summary.bru, 117
summary.bru_info, 118
summary.bru_options, 119
summary_bru, 120

toygroups, 80, 121

vertices, 122
vertices.inla.mesh-method (vertices),
 122
vertices.inla.mesh, 123