Package ‘inlabru’

December 2, 2022

Type Package
Title Bayesian Latent Gaussian Modelling using INLA and Extensions
Version 2.7.0
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)
License GPL (>= 2)
Additional_repositories https://inla.r-inla-download.org/R/testing
RoxygenNote 7.2.2
Encoding UTF-8
Depends methods, R (>= 3.6), sp (>= 1.4-5), stats
Imports MatrixModels, magrittr, Matrix, patchwork, plyr, rgdal (>=
1.5.8), rlang, sf, utils, withr, lifecycle
Suggests covr, dplyr, ggmap, ggplot2, ggpolypath, graphics, INLA (>=
21.08.31), knitr, maps, maptools, mgev, raster, RColorBrewer,
rgl, rmarkdown, scales, shiny, sn, spatstat.geom,
spatstat.data, spatstat (>= 2.0-0), sphereplot, splancs, terra,
testthat, tidyr, DiagrammeR
Config/testthat/parallel true
Config/testthat/edition 3
R topics documented:

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.robins_subset.R' 'data.seals.R'
 'data.shrimp.R' 'data.toygroungs.R' 'deltaIC.R' 'deprecated.R'
 'dsdata.R' 'dsmdata.R' 'dsmdata.tools.R' 'effect.R'
 'environment.R' 'fmesher_crs.R' 'fmesher_evaluator.R'
 'fmesher_sf_mesh.R' 'fmesher_sp_mesh.R' 'fmesher_utils.R'
 'ggplot.R' 'inla.R' 'inlabru.R' 'integration.R'
 'local_testthat.R' 'mappers.R' 'mesh.R' 'model.R' 'nlinla.R'
 'plotsample.R' 'prediction.R' 'rgl.R' 'sampling.R' 'sf_utils.R'
 'shapefile.R' 'spatstat.R' 'spde.R' 'stack.R'
 'track_plotting.R' 'transformation.R' 'util.R'

VignetteBuilder knitr

BuildVignettes true

LazyData false

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, bugfixes, and
 vignettes),
 Suen Man Ho [ctb, cph] (Man Ho Suen contributed features for aggregated
 responses and vignette updates),
 Roudier Pierre [ctb, cph] (Pierre Roudier contributed general quantile
 summaries),
 Meehan Tim [ctb, cph] (Tim Meehan contributed the SVC vignette and
 robins data)

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

Repository CRAN

Date/Publication 2022-12-02 09:50:05 UTC

R topics documented:

bincount .................................................. 5
bm_list ................................................... 6
bru ........................................................ 7
bru_call_options ........................................ 9
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>bru_compute_linearisation</td>
</tr>
<tr>
<td>bru_convergence_plot</td>
</tr>
<tr>
<td>bru_fill_missing</td>
</tr>
<tr>
<td>bru_forward_transformation</td>
</tr>
<tr>
<td>bru_get_mapper</td>
</tr>
<tr>
<td>bru_log_reset</td>
</tr>
<tr>
<td>bru_make_stack</td>
</tr>
<tr>
<td>bru_mapper</td>
</tr>
<tr>
<td>bru_mapper.generics</td>
</tr>
<tr>
<td>bru_mapper.methods</td>
</tr>
<tr>
<td>bru_options</td>
</tr>
<tr>
<td>bru_safe_inla</td>
</tr>
<tr>
<td>bru_standardise_names</td>
</tr>
<tr>
<td>bru_summarise</td>
</tr>
<tr>
<td>component</td>
</tr>
<tr>
<td>component_eval</td>
</tr>
<tr>
<td>component_list</td>
</tr>
<tr>
<td>cprod</td>
</tr>
<tr>
<td>deltaIC</td>
</tr>
<tr>
<td>devel.cvmeasure</td>
</tr>
<tr>
<td>evaluate_comp_lin</td>
</tr>
<tr>
<td>evaluate_index</td>
</tr>
<tr>
<td>evaluate_inputs</td>
</tr>
<tr>
<td>eval_spatial</td>
</tr>
<tr>
<td>expand_labels</td>
</tr>
<tr>
<td>fm_as_sp_crs</td>
</tr>
<tr>
<td>fm_CRS</td>
</tr>
<tr>
<td>fm_crs</td>
</tr>
<tr>
<td>fm_evaluate</td>
</tr>
<tr>
<td>fm_has_PROJ6</td>
</tr>
<tr>
<td>fm_spTransform</td>
</tr>
<tr>
<td>fm_sp_get_crs</td>
</tr>
<tr>
<td>fm_transform</td>
</tr>
<tr>
<td>fm_wkt_as_wkt_tree</td>
</tr>
<tr>
<td>fm_wkt_is_geocent</td>
</tr>
<tr>
<td>generate</td>
</tr>
<tr>
<td>gg</td>
</tr>
<tr>
<td>gg.data.frame</td>
</tr>
<tr>
<td>gg.inla.mesh</td>
</tr>
<tr>
<td>gg.inla.mesh.1d</td>
</tr>
<tr>
<td>gg.matrix</td>
</tr>
<tr>
<td>gg.prediction</td>
</tr>
<tr>
<td>gg.RasterLayer</td>
</tr>
<tr>
<td>gg.SpatialGridDataFrame</td>
</tr>
<tr>
<td>gg.SpatialLines</td>
</tr>
<tr>
<td>gg.SpatialPixels</td>
</tr>
<tr>
<td>gg.SpatialPixelsDataFrame</td>
</tr>
<tr>
<td>gg.SpatialPoints</td>
</tr>
<tr>
<td>R topics documented:</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>gg.SpatialPolygons</td>
</tr>
<tr>
<td>globe</td>
</tr>
<tr>
<td>glm</td>
</tr>
<tr>
<td>glm.inla.mesh</td>
</tr>
<tr>
<td>glm.SpatialLines</td>
</tr>
<tr>
<td>glm.SpatialPoints</td>
</tr>
<tr>
<td>gm</td>
</tr>
<tr>
<td>gmap</td>
</tr>
<tr>
<td>gorillas</td>
</tr>
<tr>
<td>inla.stack.mjoin</td>
</tr>
<tr>
<td>inlabru</td>
</tr>
<tr>
<td>int</td>
</tr>
<tr>
<td>ipoints</td>
</tr>
<tr>
<td>lgcp</td>
</tr>
<tr>
<td>like</td>
</tr>
<tr>
<td>mexdolphin</td>
</tr>
<tr>
<td>mrsea</td>
</tr>
<tr>
<td>multiplot</td>
</tr>
<tr>
<td>pixels</td>
</tr>
<tr>
<td>plot.bru</td>
</tr>
<tr>
<td>plot.prediction</td>
</tr>
<tr>
<td>plotsample</td>
</tr>
<tr>
<td>point2count</td>
</tr>
<tr>
<td>Poisson1_1D</td>
</tr>
<tr>
<td>Poisson2_1D</td>
</tr>
<tr>
<td>Poisson3_1D</td>
</tr>
<tr>
<td>predict.bru</td>
</tr>
<tr>
<td>robins_subset</td>
</tr>
<tr>
<td>row.kron</td>
</tr>
<tr>
<td>sample.lgcp</td>
</tr>
<tr>
<td>seals</td>
</tr>
<tr>
<td>shrimp</td>
</tr>
<tr>
<td>sline</td>
</tr>
<tr>
<td>spatial.to.ppp</td>
</tr>
<tr>
<td>spde.posterior</td>
</tr>
<tr>
<td>spoly</td>
</tr>
<tr>
<td>stransform</td>
</tr>
<tr>
<td>st_check_dim</td>
</tr>
<tr>
<td>st_check_polygon</td>
</tr>
<tr>
<td>st_signed_area</td>
</tr>
<tr>
<td>summary.bru</td>
</tr>
<tr>
<td>summary.bru_info</td>
</tr>
<tr>
<td>summary.bru_mapper</td>
</tr>
<tr>
<td>summary.bru_options</td>
</tr>
<tr>
<td>toygroups</td>
</tr>
<tr>
<td>vertices.inla.mesh</td>
</tr>
</tbody>
</table>

**Index** | 153
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 bru() or lgcp() call
predictor A formula describing the prediction of a 1D LGCP via predict().
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 predict.bru()

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)

---

**bm_list**  
*Methods for mapper lists*

**Description**

bru_mapper lists can be combined into bm_list lists.

**Usage**

```r
## S3 method for class 'bru_mapper'
c(...)  
```

```r
## S3 method for class 'bm_list'
c(...)  
```

```r
## S3 method for class 'bm_list'
x[i]  
```
Arguments

... Objects to be combined.

x bm_list object from which to extract element(s)

i indices specifying elements to extract

Methods (by generic)

• c(bm_list): The ... arguments should be bm_list objects.
• [: Extract sub-list

Functions

• c(bru_mapper): The ... arguments should be bru_mapper objects.

Examples

m <- c(A = bru_mapper_const(), B = bru_mapper_scale())
str(m)
str(m[2])

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

components  A formula-like specification of latent components. Also used to define a default linear additive predictor. See component() for details.

...  Likelihoods, each constructed by a calling like(), or named parameters that can be passed to a single like() call. Note that all the arguments will be evaluated before calling like() in order to detect if they are like objects. This means that special arguments that need to be evaluated in the context of response_data or data (such as Ntrials) may will only work that way in direct calls to like().

options  A bru_options options object or a list of options passed on to bru_options()

.envir  Environment for component evaluation (for when a non-formula specification is used)

result  A previous estimation object of class bru

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

```r
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)
```


# 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 `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:

```r
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

```r
opts <- bru_call_options()

# Print them:
```
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,
    input,
    state,
    comp_simple,
    effects,
    pred0,
    allow_latent,
    allow_combine,
    eps,
    ...
)

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

## S3 method for class 'bru_like_list'
bru_compute_linearisation(}
bru_compute_linearisation

lhoods,
model,
input,
state,
comp_simple,
eps = 1e-05,
...
)

## S3 method for class 'bru_model'
bru_compute_linearisation(model, lhoods, input, state, comp_simple, ...)

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
input Precomputed component inputs from evaluate_inputs()
state The state information, as a list of named vectors
comp_simple Component evaluation information
  • For bru_component: bru_mapper_taylor object
  • For bru_like: A comp_simple_list object for the components in the likelihood
  • For bru_like_list: A comp_simple_list_list object
effects • 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 _latent suffix.
allow_combine logical; If TRUE, the predictor expression may involve several rows of the input data to influence the same row.
esps The finite difference step size
lhood A bru_like object
lhoods A bru_like_list object
bru_convergence_plot  
*Plot inlabru convergence diagnostics*

**Description**

Draws four panels of convergence diagnostics for an iterated INLA method estimation

**Usage**

```r
bru_convergence_plot(x)
```

**Arguments**

- `x`  
  a `bru` object, typically a result from `bru()` for a nonlinear predictor model

**Examples**

```r
## Not run:
fit <- bru(...)  
bru_convergence_plot(fit)
## End(Not run)
```

---

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

**Description**

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

**Usage**

```r
bru_fill_missing(
  data,  
  where,  
  values,  
  layer = NULL,  
  selector = NULL,  
  batch_size = 50
)
```
Arguments

data A SpatialPointsDataFrame, SpatialPixelsDataFrame, SpatialGridDataFrame, SpatialRaster, Raster, or sf object containing data to use for filling

where A, matrix, data.frame, or SpatialPoints or SpatialPointsDataFrame, or sf object, containing the locations of the evaluated values

values A vector of values to be filled in where is.na(values) is TRUE

layer, selector Specifies what data column or columns from which to extract data, see component() for details.

batch_size Size of nearest-neighbour calculation blocks, to limit the memory and computational complexity.

Value

An infilled vector of values

Examples

```r
## 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$val[3, 30])
  input$val[3, 30] <- NA # Introduce missing values
  input$val <- bru_fill_missing(input, input, input$val)
  print(input$val[3, 30])
}
## End(Not run)
```
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

u <- rnorm(5, 0, 1)
y <- bru_forward_transformation(qexp, u, rate = 2)
v <- bru_inverse_transformation(pexp, y, rate = 2)
rrbind(u, y, v)

bru_get_mapper

Extract mapper information from INLA model component objects

Description

The component definitions will automatically attempt to extract mapper information from any model object by calling the generic bru_get_mapper. Any class method implementation should return a bru_mapper object suitable for the given latent model.
Usage

bru_get_mapper(model, ...)

## S3 method for class 'inla.spde'
bru_get_mapper(model, ...)

## S3 method for class 'inla.rgeneric'
bru_get_mapper(model, ...)

bru_get_mapper_safely(model, ...)

Arguments

model A model component object
...
Arguments passed on to other methods

Details

• bru_get_mapper.inla.spde extract an indexed mapper for the model$mesh object contained in the model object. It returns NULL gives a warning if no known mesh type is found in the model object.

• bru_get_mapper.inla.rgeneric returns the mapper given by a call to model$f$rgeneric$definition("mapper"). To support this for your own inla.rgeneric models, add a "mapper" option to the cmd argument of your rgeneric definition function. You will need to store the mapper in your object as well. Alternative, define your model using a subclass and define a corresponding bru_get_mapper.subclass method that should return the corresponding bru_mapper object.

• bru_get_mapper_safely tries to call the bru_get_mapper, and returns NULL if it fails (e.g. due to no available class method). If the call succeeds and returns non-NULL, it checks that the object inherits from the bru_mapper class, and gives an error if it does not.

Value

A bru_mapper object defined by the model component

See Also

bru_mapper for mapper constructor methods, and bru_mapper_methods for method generics and specific implementations.

Examples

if (bru_safe_inla(quietly = TRUE)) {
  library(INLA)
  mesh <- inla.mesh.create(globe = 2)
  spde <- inla.spde2.pcmatern(mesh,
    prior.range = c(1, 0.5),
    prior.sigma = c(1, 0.5)
bru_log_reset

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 cat(...). If FALSE, return the raw log as a vector of strings, suitable for cat(..., sep = "\n"). Default: FALSE
...	Zero or more objects passed on to base::makeMessage()
domain     Domain for translations, passed on to base::makeMessage()
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 message(txt). Default: bru_options_get("bru_verbose")
verbose_store Same as `verbose`, but controlling what messages are stored in the global log object. Can be controlled via the `bru_verbose_store` with `bru_options_set()`.

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**

```r
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(lhhoods, lin, idx, ...)
```

**Arguments**

- `...`  
  Arguments passed on to other methods
- `lhood`  
  A `bru_like` object
- `lin`  
  Linearisation information
  - For `.bru_like`, a `bru_mapper_taylor` object
  - For `.bru_like_list`, a list of `bru_mapper_taylor` objects
- `idx`  
  Output from `evaluate_index(...)`
- `lhhoods`  
  A `bru_like_list` object

bru_mapper  

*Constructors for bru_mapper objects*

**Description**

Constructors for bru_mapper objects

**Usage**

```r
bru_mapper(...)  

bru_mapper_define(mapper, new_class = NULL, ..., methods = NULL)

## Default S3 method:
bru_mapper(...)

## S3 method for class `inla.mesh`
```

```
bru_mapper

bru_mapper(mesh, ...)

## S3 method for class 'inla.mesh.1d'
bru_mapper(mesh, indexed = NULL, ...)

bru_mapper_index(n = 1L, ...)

bru_mapper_taylor(
  offset = NULL,
  jacobian = NULL,
  state0 = NULL,
  ...,  
  values_mapper = NULL
)

bru_mapper_linear(...)

bru_mapper_matrix(labels, ...)

bru_mapper_factor(values, factor_mapping, indexed = FALSE, ...)

bru_mapper_const(...)

bru_mapper_scale(mapper = NULL, ...)

bru_mapper_aggregate(rescale = FALSE, n_block = NULL, ...)

bru_mapper_logsumexp(rescale = FALSE, n_block = NULL, ...)

bru_mapper_pipe(mappers, ...)

bru_mapper_multi(mappers, ...)

bru_mapper_collect(mappers, hidden = FALSE, ...)

bru_mapper_harmonics(
  order = 1,
  scaling = 1,
  intercept = TRUE,
  interval = c(0, 1),
  ...
)

Arguments

... Deprecated, alternative way to supply optional method definitions.

mapper For bru_mapper_define, a prototype mapper object, see Details. For bru_mapper_scale, a mapper to be scaled.
new_class
If non-NULL, this is added at the front of the class definition
methods
Deprecated.
mesh
An inla.mesh.1d or inla.mesh.2d object to use as a mapper
indexed
logical; if TRUE, the ibm_values() method will return an integer vector instead of the factor levels. This is needed e.g. for group and replicate mappers, since INLA::f() doesn’t accept factor values. Default: FALSE, which works for the main input mappers. The default mapper constructions will set it the required setting.
n
Size of a model for bru_mapper_index
offset
For bru_mapper_taylor, an offset vector evaluated at state0. May be NULL, interpreted as an all-zero vector of length determined by a non-null Jacobian.
jacobian
For bru_mapper_taylor, the Jacobian matrix, evaluated at state0, or a named list of such matrices. May be NULL or an empty list, for a constant mapping.
state0
For bru_mapper_taylor, the state the linearisation was evaluated at, or a list of length matching the jacobian list. NULL is interpreted as 0.
values_mapper
mapper object to be used for ibm_n and ibm_values for inla_f=TRUE (experimental, currently unused)
labels
Column labels for matrix mappings
values
Input values calculated by input_eval.bru_input()
factor_mapping
character; selects the type of factor mapping.
  • 'contrast' for leaving out the first factor level.
  • 'full' for keeping all levels.
rescale
logical; For bru_mapper_aggregate and bru_mapper_logsumexp, specifies if the blockwise sums should be normalised by the blockwise weight sums or not:
  • FALSE: (default) Straight weighted sum, no rescaling.
  • TRUE: Divide by the sum of the weight values within each block. This is useful for integration averages, when the given weights are plain integration weights. If the weights are NULL or all ones, this is the same as dividing by the number of entries in each block.
n_block
Predetermined number of output blocks. If NULL, overrides the maximum block index in the inputs.
mappers
A list of bru_mapper objects
hidden
logical, set to TRUE to flag that the mapper is to be used as a first level input mapper for INLA::f() in a model that requires making only the first mapper visible to INLA::f() and INLA::inla.stack(), such as for "bym2" models, as activated by the inla_f argument to ibm_n, ibm_values, and ibm_jacobian. Set to FALSE to always access the full mapper, e.g. for rgeneric models
order
For bru_mapper_harmonics, specifies the maximum cos/sin order. (Default 1)
scaling
For bru_mapper_harmonics, specifies an optional vector of scaling factors of length intercept + order, or a common single scalar.
intercept
logical; For bru_mapper_harmonics, if TRUE, the first basis function is a constant. (Default TRUE)
interval
numeric length-2 vector specifying a domain interval. Default c(0, 1).
bru_mapper

Methods (by class)

- **bru_mapper**(default): Calls bru_mapper_define, passing all arguments along. Mapper implementations should call bru_mapper_define() instead, and supply at least a new_class class name. Use of the bru_mapper.default method will be deprecated from version 2.7.0.
- **bru_mapper**(inla.mesh): Creates a mapper for 2D inla.mesh objects
- **bru_mapper**(inla.mesh.1d): Create mapper for an inla.mesh.1d object

Functions

- **bru_mapper()**: Generic mapper S3 constructor, used for constructing mappers for special objects. See below for details of the default constructor bru_mapper_define() that can be used to define new mappers in user code.
- **bru_mapper_define()**: Adds the new_class and "bru_mapper" class names to the inheritance list for the input mapper object, unless the object already inherits from these.

To register mapper classes and methods in scripts, use .S3method() to register the methods, e.g. .S3method("ibm_jacobian", "my_mapper_class", ibm_jacobian.my_mapper_class).

In packages with Suggests: inlabru, add method information for delayed registration, e.g.:

```
# @rawNamespace S3method(inlabru::bru_get_mapper, inla_rspde)
# @rawNamespace S3method(inlabru::ibm_n, bru_mapper_inla_rspde)
# @rawNamespace S3method(inlabru::ibm_values, bru_mapper_inla_rspde)
# @rawNamespace S3method(inlabru::ibm_jacobian, bru_mapper_inla_rspde)
```

or before each method, use @exportS3Method:

```
# @exportS3Method inlabru::bru_get_mapper
e tc., which semi-automates it.
```

- **bru_mapper_index()**: Create a an indexing mapper
- **bru_mapper_taylor()**: Provides a pre-computed affine mapping, internally used to represent and evaluate linearisation information. The state0 information indicates for which state the offset was evaluated; The affine mapper output is defined as effect(state) = offset + jacobian %*% (state - state0)
- **bru_mapper_linear()**: Create a mapper for linear effects
- **bru_mapper_matrix()**: Create a matrix mapper, for a given number of columns
- **bru_mapper_factor()**: Create a factor mapper
- **bru_mapper_const()**: Create a constant mapper
- **bru_mapper_scale()**: Create a standalone scaling mapper that can be used as part of a bru_mapper_pipe. If mapper is non-null, the bru_mapper_scale() constructor returns bru_mapper_pipe(list(mapper = mapper, scale = bru_mapper_scale()))
- **bru_mapper_aggregate()**: Constructs a mapper that aggregates elements of the input state, so it can be used e.g. for weighted summation or integration over blocks of values.
- **bru_mapper_logsumexp()**: Constructs a mapper that aggregates elements of exp(state), with optional non-negative weighting, and then takes the log(), so it can be used e.g. for \( v_k = \log(\sum_{i \in I_k} w_i \exp(u_i)) \) and \( \bar{v}_k = \log(\frac{\sum_{i \in I_k} w_i \exp(u_i)}{\sum_{i \in I_k} w_i}) \) calculations. Relies on the input handling methods for bru_mapper_aggregate, but also allows the weights to be
supplied on a logarithmic scale as \( \log_{\text{weights}} \). To avoid numerical overflow, it uses the common method of internally shifting the state blockwise with 
\[
(state - \log_{\text{weights}})[\text{block}] - \max((state - \log_{\text{weights}})[\text{block}]),
\]
and shifting the result back afterwards.

- **bru_mapper_pipe()**: Create a pipe mapper, where, mappers is a list of mappers, where the evaluated output of each mapper is handed as the state to the next mapper.. The input format for the \texttt{ibm_eval} and \texttt{ibm_jacobian} methods is a list of inputs, one for each mapper.

- **bru_mapper_multi()**: Constructs a rowwise Kronecker product mapping

- **bru_mapper_collect()**: Constructs a concatenated collection mapping

- **bru_mapper_harmonics()**: Constructs a mapper for \( \cos/\sin \) functions of orders 1 (if intercept is \texttt{TRUE}, otherwise 0) through order. The total number of basis functions is \( \text{intercept} + 2 \times \text{order} \).

  Optionally, each order can be given a non-unit scaling, via the scaling vector, of length \( \text{intercept} + \text{order} \). This can be used to give an effective spectral prior. For example, let

  \[
  \text{scaling} = 1 / (1 + (0:4)^2)
  \]

  \[
  x <- \text{seq}(0, 1, \text{length.out} = 11)
  \]

  \[
  \text{bmh1} = \text{bru_mapper_harmonics}(\text{order} = 4, \text{interval} = \text{c}(0, 1))
  \]

  \[
  \text{u1} \leftarrow \text{ibm_eval}(
  \text{bmh1,}
  \text{input} = x,
  \text{state} = \text{rnorm}(9, \text{sd} = \text{rep(\text{scaling}, \text{c}(1, 2, 2, 2, 2)))}
  \)
  \]

  Then, with

  \[
  \text{bmh2} = \text{bru_mapper_harmonics}(\text{order} = 4, \text{scaling} = \text{scaling})
  \]

  \[
  \text{u2} = \text{ibm_eval}(\text{bmh2, input} = x, \text{state} = \text{rnorm}(9))
  \]

  the stochastic properties of \( \text{u1} \) and \( \text{u2} \) will be the same, with \( \text{scaling}^2 \) determining the variance for each frequency contribution.

  The period for the first order harmonics is shifted and scaled to match \text{interval}.

**See Also**

- **bru_mapper_generics** for generic methods, **bru_mapper_methods** for specific method implementations, and **bru_get_mapper** for hooks to extract mappers from latent model object class objects.

**Examples**

```r
mapper <- bru_mapper_index(5)
ibm_jacobian(mapper, input = c(1, 3, 4, 5, 2))
```
bru_mapper_generics

Generic methods for bru_mapper objects

Description

A bru_mapper sub-class implementation must provide an `ibm_jacobian()` 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

```r
ibm_n(mapper, inla_f = FALSE, ...)
ibm_n_output(mapper, input, state = NULL, inla_f = FALSE, ...)
ibm_values(mapper, inla_f = FALSE, ...)
ibm_amatrix(mapper, input, state = NULL, inla_f = FALSE, ...)
ibm_is_linear(mapper, ...)
ibm_jacobian(mapper, input, state = NULL, inla_f = FALSE, ...)
ibm_linear(mapper, input, state = NULL, ...)
ibm_eval(mapper, input, state = NULL, ...)
ibm_names(mapper)
ibm_names(mapper) <- value
ibm_inla_subset(mapper, ...)
ibm_invalid_output(mapper, input, state, ...)
```

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

## Default S3 method:
```r
ibm_n_output(mapper, input, state = NULL, inla_f = FALSE, ...)
```

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

## Default S3 method:
```r
ibm_amatrix(mapper, ...)
```
## Default S3 method:
ibm_is_linear(mapper, ...)

## Default S3 method:
ibm_jacobian(mapper, input, state, ...)

## Default S3 method:
ibm_linear(mapper, input, state, ...)

## Default S3 method:
ibm_eval(mapper, input, state = NULL, ...)

## Default S3 method:
ibm_names(mapper, ...)

## Default S3 method:
ibm_inla_subset(mapper, ...)

## Default S3 method:
ibm_invalid_output(mapper, input, state, ...)

### Arguments

- **mapper**
  - A mapper S3 object, inheriting from bru_mapper.

- **inla_f**
  - logical; when TRUE for ibm_n() and ibm_values(), the result must be compatible with the INLA::f(...) and corresponding INLA::inla.stack(...) constructions. For ibm_(eval,jacobian,linear), the input interpretation may be different. 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 bru_mapper_collect.

- **...**
  - Arguments passed on to other methods

- **input**
  - Data input for the mapper.

- **state**
  - A vector of latent state values for the mapping, of length ibm_n(mapper, inla_f = FALSE)

- **value**
  - a character vector of the same length as the number of sub-mappers in the mapper

### Functions

- **ibm_n()**: Implementations must return the size of the latent vector being mapped to.

- **ibm_n_output()**: Implementations must return an integer denoting the mapper output length. The default implementation returns NROW(input). Mappers such as bru_mapper_multi and bru_mapper_collect, that can accept list() inputs require their own methods implementations.

- **ibm_values()**: When inla_f=TRUE, 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()` will become deprecated in 2.7.0. Use `ibm_jacobian` instead. Implementations must return a (sparse) matrix of size `ibm_n_output(…)` by `ibm_n(…)`. The `inla_f=TRUE` argument should only affect the allowed type of input format.
- `ibm_is_linear()` Implementations must return `TRUE` or `FALSE`. If `TRUE` (returned by the default method unless the mapper contains an `is_linear` variable), users of the mapper may assume the mapper is linear.
- `ibm_jacobian()` Implementations must return a (sparse) matrix of size `ibm_n_output(mapper, input, inla_f)` by `ibm_n(mapper, inla_f = FALSE)`. The `inla_f=TRUE` argument should only affect the allowed type of input format.
- `ibm_linear()` Implementations must return an `bru Mapper_taylor` object. The linearisation information includes `offset`, `jacobian`, and `state0`. The state information indicates for which state the offset was evaluated, with `NULL` meaning all-zero. The linearised mapper output is defined as `effect(input, state) = offset(input, state0) + Jacobian(input, state0) %*% (state - state0)`. The default method calls `ibm_eval()` and `ibm_jacobian()` to generate the needed information.
- `ibm_eval()` Implementations must return a vector of length `ibm_n_output(…)`. The input contents must be in a format accepted by `ibm_jacobian(…)` for the mapper.
- `ibm_names()` Implementations must return a character vector of sub-mapper names, or `NULL`. Intended for providing information about multi-mappers and mapper collections.
- `ibm_names(mapper) <- value` Set mapper names.
- `ibm_inla_subset()` 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_invalid_output()` Implementations should return a logical vector of length `ibm_n_output(mapper, input, state, …)` indicating which, if any, output elements of `ibm_eval(mapper, input, state, …)` are known to be invalid. For for multi/collect mappers, a list, when given a multi=TRUE argument.
- `ibm_n(default)` 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.
- `ibm_n_output(default)` Returns `NROW(input)`
- `ibm_values(default)` Returns a non-null element 'values' from the mapper object, and `seq_len(ibm_n(mapper))` if it doesn't exist.
- `ibm_amatrix(default)` Gives an error message. Mapper classes must implement their own `ibm_jacobian` or `ibm_amatrix` methods. New implementations should use a `ibm_jacobian` method. `ibm_amatrix` may become deprecated in a future version.
- `ibm_is_linear(default)` Returns logical `is_linear` from the mapper object if it exists, and otherwise `TRUE`.
- `ibm_jacobian(default)` Calls `ibm_amatrix`, which by default gives an error. Mapper classes should implement their own `ibm_jacobian` method.
- `ibm_linear(default)` Calls `ibm_eval()` and `ibm_jacobian()` and returns a `bru Mapper_taylor` object. The state0 information in the affine mapper indicates for which state the offset was evaluated; The affine mapper output is defined as `effect(input, state) = offset(input, state0) + Jacobian(input, state0) %*% (state - state0)`.
• ibm_eval(default): Verifies that the mapper is linear with ibm_is_linear(), and then computes a linear mapping as ibm_jacobian(...) %*% state. When state is NULL, a zero vector of length ibm_n_output(...) is returned.
• ibm_names(default): Returns NULL
• ibm_inla_subset(default): 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).
• ibm_invalid_output(default): Returns an all-FALSE logical vector.

See Also
bru_mapper for constructor methods, and bru_get_mapper for hooks to extract mappers from latent model object class objects.
bru_mapper, bru_mapper_methods

Examples

# ibm_names
mapper <- bru_mapper_multi(list(
  A = bru_mapper_index(2),
  B = bru_mapper_index(2)
))
ibm_names(mapper)
ibm_names(mapper) <- c("new", "names")
ibm_names(mapper)

 bru_mapper_methods    Methods for bru_mapper objects

Description
A bru_mapper sub-class implementation must provide an ibm_jacobian() 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

## 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_jacobian(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_jacobian(mapper, input, ...)

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

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

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

## S3 method for class 'bru_mapper_taylor'
ibm_n_output(mapper, input, ...)

## S3 method for class 'bru Mapper index'
ibm_values(mapper, inla_f = FALSE, multi = FALSE, ...)

## S3 method for class 'bru Mapper index'
ibm_jacobian(mapper, input, state, ...)

ibm_eval(mapper, input = NULL, state = NULL, ...)

## 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_jacobian(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_jacobian(mapper, input, state = NULL, inla_f = FALSE, ...)

## 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_jacobian(mapper, input, ...)
## S3 method for class 'bru_mapper_const'
ibm_n(mapper, ...)
## S3 method for class 'bru_mapper_const'
ibm_values(mapper, ...)
## S3 method for class 'bru_mapper_const'
ibm_jacobian(mapper, input, ...)
## S3 method for class 'bru_mapper_const'
ibm_eval(mapper, input, state = NULL, ...)
## S3 method for class 'bru_mapper_scale'
ibm_n(mapper, ..., state = NULL, n_state = NULL)
## S3 method for class 'bru_mapper_scale'
ibm_n_output(mapper, input, state = NULL, ..., n_state = NULL)
## S3 method for class 'bru_mapper_scale'
ibm_values(mapper, ..., state = NULL, n_state = NULL)
## S3 method for class 'bru_mapper_scale'
ibm_jacobian(mapper, input, state = NULL, ..., sub_lin = NULL)
## S3 method for class 'bru_mapper_scale'
ibm_linear(mapper, input, state, ...)
## S3 method for class 'bru_mapper_scale'
ibm_eval(mapper, input, state = NULL, ..., sub_lin = NULL)
## S3 method for class 'bru_mapper_aggregate'
ibm_n(mapper, ..., input = NULL, state = NULL, n_state = NULL)
## S3 method for class 'bru_mapper_aggregate'
ibm_n_output(mapper, input = NULL, ...)
## S3 method for class 'bru_mapper_aggregate'
ibm_values(mapper, ..., state = NULL, n_state = NULL)
## S3 method for class 'bru_mapper_aggregate'
ibm_jacobian(mapper, input, state = NULL, ...)

## S3 method for class 'bru_mapper_aggregate'
ibm_eval(mapper, input, state = NULL, ..., sub_lin = NULL)

## S3 method for class 'bru_mapper_aggregate'
ibm_linear(mapper, input, state, ...)

## S3 method for class 'bru_mapper_logsumexp'
ibm_jacobian(mapper, input, state = NULL, ...)

## S3 method for class 'bru_mapper_logsumexp'
ibm_eval(mapper, input, state = NULL, ..., sub_lin = NULL)

## S3 method for class 'bru_mapper_logsumexp'
ibm_linear(mapper, input, state, ...)

## S3 method for class 'bru_mapper_pipe'
ibm_n(mapper, ..., state = NULL)

## S3 method for class 'bru_mapper_pipe'
ibm_n_output(mapper, input, state = NULL, ...)

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

## S3 method for class 'bru_mapper_pipe'
ibm_jacobian(mapper, input, state = NULL, ...)

## S3 method for class 'bru_mapper_pipe'
ibm_linear(mapper, input, state, ...)

## S3 method for class 'bru_mapper_pipe'
ibm_eval(mapper, input, state = NULL, ...)

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

## S3 method for class 'bru_mapper_multi'
ibm_n_output(mapper, input, ...)

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

## S3 method for class 'bru_mapper_multi'
ibm_is_linear(mapper, multi = FALSE, ...)

## S3 method for class 'bru_mapper_multi'
ibm_jacobian(
mapper,
input,
state = NULL,
inla_f = FALSE,
multi = FALSE,
..., 
sub_A = NULL 
)

## S3 method for class 'bru_mapper_multi'
ibm_linear(mapper, input, state, inla_f = FALSE, ...)

## S3 method for class 'bru_mapper_multi'
ibm_eval(mapper, input, state = NULL, inla_f = FALSE, ...)

## S3 method for class 'bru_mapper_multi'
ibm_invalid_output(mapper, input, state, inla_f = FALSE, multi = FALSE, ...)

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

## S3 method for class 'bru_mapper_multi'
ibm_names(mapper)

## S3 replacement method for class 'bru_mapper_multi'
ibm_names(mapper) <- value

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

## S3 method for class 'bru_mapper_collect'
ibm_n_output(mapper, input, state = NULL, inla_f = FALSE, multi = FALSE, ...)

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

## S3 method for class 'bru_mapper_collect'
ibm_is_linear(mapper, inla_f = FALSE, multi = FALSE, ...)

## S3 method for class 'bru_mapper_collect'
ibm_jacobian(
mapper,
input,
state = NULL,
inla_f = FALSE,
multi = FALSE,
..., 
sub_A = NULL 
)
sub_lin = NULL

## S3 method for class 'bru_mapper_collect'
ibm_eval(
  mapper,
  input,
  state,
  inla_f = FALSE,
  multi = FALSE,
  ...
)

## S3 method for class 'bru_mapper_collect'
ibm_linear(mapper, input, state, inla_f = FALSE, ...)

## S3 method for class 'bru_mapper_collect'
ibm_invalid_output(mapper, input, state, inla_f = FALSE, multi = FALSE, ...)

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

## S3 method for class 'bru_mapper_collect'
ibm_names(mapper)

## S3 replacement method for class 'bru_mapper_collect'
ibm_names(mapper) <- value

## S3 method for class 'bru_mapper_harmonics'
ibm_n(mapper, inla_f = FALSE, ...)

## S3 method for class 'bru_mapper_harmonics'
ibm_jacobian(mapper, input, state = NULL, inla_f = FALSE, ...)

Arguments

mapper A mapper S3 object, inheriting from bru_mapper.
...
Arguments passed on to other methods
input Data input for the mapper.
state A vector of latent state values for the mapping, of length ibm_n(mapper, inla_f = FALSE)
inla_f logical; when TRUE for ibm_n() and ibm_values(), the result must be compatible with the INLA::f(...) and corresponding INLA::inla.stack(...) constructions. For ibm_{eval,jacobian,linear}, the input interpretation may be different. 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 bru_mapper_collect.
**bru_mapper_methods**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>multi</td>
<td>logical; If TRUE (or positive), recurse one level into sub-mappers</td>
</tr>
<tr>
<td>n_state</td>
<td>integer giving the length of the state vector for mappers that have state dependent output size.</td>
</tr>
<tr>
<td>sub_lin</td>
<td>Internal, optional pre-computed sub-mapper information</td>
</tr>
<tr>
<td>sub_A</td>
<td>Internal; precomputed Jacobian matrices.</td>
</tr>
<tr>
<td>pre_A</td>
<td>Internal; precomputed Jacobian matrix.</td>
</tr>
<tr>
<td>x</td>
<td>object from which to extract element(s)</td>
</tr>
<tr>
<td>i</td>
<td>indices specifying element(s) to extract</td>
</tr>
<tr>
<td>drop</td>
<td>logical; For [[.bru_mapper_collect, whether to extract an individual mapper when i identifies a single element. If FALSE, a list of sub-mappers is returned (suitable e.g. for creating a new bru_mapper_collect object). Default: TRUE</td>
</tr>
<tr>
<td>value</td>
<td>a character vector of up to the same length as the number of mappers in the multi-mapper x</td>
</tr>
</tbody>
</table>

**Details**

- The `ibm_eval.bru_mapper_taylor()` evaluates linearised mapper information at the given state. The input argument is ignored, so that the usual argument order `ibm_eval(mapper, input, state)` syntax can be used, but also `ibm_eval(mapper, state = state)`. For a mapper with a named jacobian list, the state argument must also be a named list. If state is `NULL`, all-zero is assumed.

For `bru_mapper_scale`, input values without a scale element are interpreted as no scaling.

- For `bru mapper_aggregate`, input should be a list with elements block and weights. block should be a vector of the same length as the state, or `NULL`, with `NULL` equivalent to all-1. If weights is `NULL`, it’s interpreted as all-1.

- For `bru_mapper_logsumexp`, input should be a list with elements block and weights. block should be a vector of the same length as the state, or `NULL`, with `NULL` equivalent to all-1. If weights is `NULL`, it’s interpreted as all-1.

- `ibm_jacobian` 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 `ibm_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_invalid_output` 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 `ibm_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_jacobian` 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 `ibm_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.
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_invalid_output 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 ibm_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

- \([\cdot \text{indexing a } \text{bru_mapper_multi} \text{ extracts a subset } \text{bru_mapper_multi} \text{ object (for drop } \text{FALSE}) \text{ or an individual sub-mapper (for drop } \text{TRUE}, \text{ and } i \text{ identifies a single element)}

- \([\cdot \text{indexing a } \text{bru_mapper_collect} \text{ extracts a subset } \text{bru_mapper_collect} \text{ object (for drop } \text{FALSE}) \text{ or an individual sub-mapper (for drop } \text{TRUE}, \text{ and } i \text{ identifies a single element)}

- The names() method for bru_mapper_collect returns the names from the sub-mappers list

Functions

- ibm_names(bru_mapper_multi): Returns the names from the sub-mappers list

See Also

bru_mapper for constructor methods, and bru_get_mapper for hooks to extract mappers from latent model object class objects.

bru_mapper, bru_mapper_generics

---

**bru_options**

Create or update an options objects

**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

...  A collection of named options, optionally including one or more bru_options objects. Options specified later override the previous options.
x  An object to be converted to an bru_options object.
options  An bru_options object to be checked
ignore_null  Ignore missing or NULL options.
name  Either NULL, or single option name string, or character vector or list with option names. Default: NULL
include_default  logical; If TRUE, the default options are included together with the global override options. Default: TRUE
.reset  For bru_options_set, 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 ≤ 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 ≤ 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)
  - **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 \(\text{mean}(\eta)\). 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

```r
## 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)
}
```

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

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

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

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

```r
## 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

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

quietly 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)
bru_summarise

Arguments

x character vector; names to be standardised

Value

A character vector with standardised names

Examples

bru_standardise_names("Precision for the Gaussian observations")

bru_summarise data, probs = c(0.025, 0.5, 0.975), x = NULL, cbind.only = FALSE, max_moment = 2

Arguments

data A list of samples, each either numeric or a data.frame
probs A numeric vector of probabilities with values in [0, 1], passed to stats::quantile
x A data.frame of data columns that should be added to the summary data frame
cbind.only If TRUE, only cbind the samples and return a matrix where each column is a sample
max_moment integer, at least 2. Determines the largest moment order information to include in the output. If max_moment > 2, includes "skew" (skewness, \(E[(x-m)^3/s^3]\)), and if max_moment > 3, includes "ekurtois" (excess kurtosis, \(E[(x-m)^4/s^4] - 3\)). Default 2. Note that the Monte Carlo variability of the ekurtois estimate may be large.

Value

A data.frame or Spatial[Points/Pixels]DataFrame with summary statistics, "mean", "sd", paste0("q", probs), "mean.mc_std_err", "sd.mc_std_err"
Examples

bru_summarise(matrix(rexp(10000), 10, 1000), max_moment = 4, probs = NULL)

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 "const" (same as "offset"), "factor_full", "factor_contrast", "linear", "fixed", or a model name or object accepted by INLA’s f function. If set to NULL, then "linear" 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

The _layer input should evaluate to a numeric index or character name or vector of which layer/variable to extract from a covariate data object given in main. (Default: NULL if _selector is given. Otherwise the effect component name, if it exists in the covariate object, and otherwise the first column of the covariate data frame)

The _selector value should be a character name of a variable whose contents determines which layer to extract from a covariate for each data point. (Default: NULL)

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)
component

```

| copy      | character: label of other component that this component should be a copy of. If the fixed = FALSE, a scaling constant is estimated, via a hyperparameter. If fixed = TRUE, 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 ~ 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 $\eta$ would be written down as

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

where:

- $c$ is the intercept
- $x$ is a covariate
- $\beta$ is a latent variable associated with $x$ and
- $\psi = \beta \ast x$ is called the 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 $\psi = \beta \ast x$ as $x$. Hence, it is not clear when $x$ 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 \( \psi \) 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 \( e^{\beta x + c} \).

On the other hand, predict may be used to only look at a transformation of the latent variable \( \beta \psi \)

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

which corresponds to the mathematical expression \( e^{\beta} \).

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**

```r
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
## 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(z = rnorm(5))
  )
  fit <- bru(z ~ -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)

---

### component_list

**Methods for inlabru component lists**

#### Description

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

#### Usage

```r
component_list(object, lhoods = NULL, .envir = parent.frame(), ...)
```

---

## S3 method for class 'formula'

```r
component_list(object, lhoods = NULL, .envir = parent.frame(), ...)
```

---

## S3 method for class 'list'

```r
component_list(object, lhoods = NULL, .envir = parent.frame(), ...)
```

---

## S3 method for class 'component_list'

```r
c(...)
```

---

## S3 method for class 'component'

c(...)  

```
## S3 method for class 'component_list'
```

x[i]

**Arguments**

- **object**
  - The object to operate on
- **lhoods**
  - A bru_like_list object
- **.envir**
  - An evaluation environment for non-formula input
- **...**
  - Parameters passed on to other methods. Also see Details.
- **x**
  - A component_list object from which to extract a sub-list
- **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 component_list.
- **c.component**: The ... arguments should be component objects. The environment from the first argument will be applied to the resulting "component_list".

**Author(s)**

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

**See Also**

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 lgcpl objects.*

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

**Usage**

```r
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**

```r
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

Variance and correlations measures for prediction components

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 the first component.
- prediction2: A prediction of the second 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

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)

```r
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()

fit <- lgcp(cmp, gorillas.nests,
  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.inla.mesh(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"),}
```

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
```
evaluate_comp_lin

Compute all component linearisations

Description
Computes individual bru_mapper_taylor objects for included components for each model likelihood

Usage
evaluate_comp_lin(model, input, state, inla_f = FALSE)

Arguments
- model: A bru_model object
- input: A list of named lists of component inputs
- state: A named list of component states
- inla_f: Controls the input data interpretations

Value
A list (class 'comp_simple') of named lists (class 'comp_simple_list') of bru_mapper_taylor objects, one for each included component

evaluate_index
Compute all index values

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.

---

**evaluate_inputs**

*Compute all component inputs*

**Description**

Computes the component inputs for included components for each model likelihood

**Usage**

```r
evaluate_inputs(model, lhoods, inla_f)
```

**Arguments**

- `model`: A `bru_model` object
- `lhoods`: A `bru_like_list` object
- `inla_f`: logical

---

**eval_spatial**

*Evaluate spatial covariates*

**Description**

Evaluate spatial covariates

**Usage**

```r
eval_spatial(data, where, layer = NULL, selector = NULL)
```
expand_labels

Arguments

- **data** Spatial grid-like data
- **where** Where to evaluate the data
- **layer** Which data layer to extract (as integer or character). May be a vector, specifying a separate layer for each where point.
- **selector** The name of a variable in where specifying the layer information.

---

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_as_sp_crs

Coercion methods to and from meshes

Description

Coercion methods to and from meshes
**Usage**

```r
fm_as_sp_crs(x, ...)

fm_as_sfc(x)

fm_as_inla_mesh_segment(...)

fm_as_inla_mesh(...)
```

```r
## S3 method for class 'Var'
fm_as_sfc(x, ...)

## S3 method for class 'Var'
sfc_MULTIPOLYGON

## S3 method for class 'Var'
sfc_POINT

## S3 method for class 'Var'
sfc_LINESTRING

## S3 method for class 'Var'
sfc_POLYGON

## S3 method for class 'Var'
sf

## S3 method for class 'Var'
sf

## S3 method for class 'Var'
sf

## S3 method for class 'Var'
sf

fm_sp2segment(sp, ...)

## S3 method for class 'matrix'
fm_as_inla_mesh_segment(
  sp,
  reverse = FALSE,
  grp = NULL,
  is.bnd = FALSE,
  crs = NULL,
  closed = FALSE,
  ...
)

## S3 method for class 'SpatialPoints'
fm_as_inla_mesh_segment(
  sp,
  reverse = FALSE,
  grp = NULL,
  ...)
```
## S3 method for class 'SpatialPointsDataFrame'
fm_as_inla_mesh_segment(sp, ...)

## S3 method for class 'Line'
fm_as_inla_mesh_segment(sp, reverse = FALSE, grp = NULL, crs = NULL, ...)

## S3 method for class 'Lines'
fm_as_inla_mesh_segment(sp, join = TRUE, grp = NULL, crs = NULL, ...)

## S3 method for class 'SpatialLines'
fm_as_inla_mesh_segment(sp, join = TRUE, grp = NULL, ...)

## S3 method for class 'SpatialLinesDataFrame'
fm_as_inla_mesh_segment(sp, ...)

## S3 method for class 'SpatialPolygons'
fm_as_inla_mesh_segment(sp, join = TRUE, grp = NULL, ...)

## S3 method for class 'SpatialPolygonsDataFrame'
fm_as_inla_mesh_segment(sp, ...)

## S3 method for class 'Polygons'
fm_as_inla_mesh_segment(sp, join = TRUE, crs = NULL, grp = NULL, ...)

## S3 method for class 'Polygon'
fm_as_inla_mesh_segment(sp, crs = NULL, ...)

### Arguments

x       An object to be coerced/transformed/converted into another class
...
Arguments passed on to other methods

reverse  logical; When TRUE, reverse the order of the input points. Default FALSE

grp      if non-null, should be an integer vector of grouping labels for one for each segment. Default NULL

is.bnd   logical; if TRUE, set the boundary flag for the segments. Default TRUE

join     logical; if TRUE, join input segments with common vertices. Default TRUE

sp       An sp style S4 object to be converted

crs      A crs object

closed   logical; whether to treat a point sequence as a closed polygon. Default: FALSE
Value

- `fm_as_sfc`: An `sfc.MULTIPOLYGON` object
- `fm_as_inla_mesh`: An `inla.mesh` mesh object

---

Value

- `fm_as_sfc`: An `sfc.MULTIPOLYGON` object
- `fm_as_inla_mesh`: An `inla.mesh` mesh object

---

**Description**

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

**Usage**

```r
fm_CRS(...)  
```

```r
## S3 method for class 'crs'
fm_CRS(x, ...)
```

```r
## S3 method for class 'fm_crs'
fm_CRS(x, ...)
```

```r
## S3 method for class 'Spatial'
fm_CRS(x, ...)
```

```r
## S3 method for class 'inla.CRS'
fm_CRS(x, ..., crsonly = FALSE)
```

```r
## S3 method for class 'sf'
fm_CRS(x, ..., crsonly = FALSE)
```

```r
## S3 method for class 'sfc'
fm_CRS(x, ..., crsonly = FALSE)
```

```r
## S3 method for class 'sfg'
fm_CRS(x, ..., crsonly = FALSE)
```

```r
## S3 method for class 'inla.mesh'
fm_CRS(x, ..., crsonly = FALSE)
```

```r
## S3 method for class 'inla.mesh.lattice'
fm_CRS(x, ..., crsonly = FALSE)
```

```r
## S3 method for class 'inla.segment'
fm_CRS(x, ..., crsonly = FALSE)
```

```r
## S3 method for class 'CRS'
```
fm_CRS(x, oblique = NULL, ...)

## Default S3 method:
fm_CRS(
  projargs = NULL,
  doCheckCRSArgs = TRUE,
  args = NULL,
  oblique = NULL,
  SRS_string = NULL,
  ...
)

Arguments

\begin{itemize}
  \item \ldots Additional parameters. Not currently in use.
  \item \texttt{x} Object to convert to CRS or to extract CRS information from.
  \item \texttt{crsonly} logical; if \texttt{TRUE}, remove any oblique information for \code{inla.CRS} class objects and return a \code{pureCrs} object.
  \item \texttt{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.
  \item \texttt{projargs} Either 1) a projection argument string suitable as input to \code{sp::CRS}, or 2) an existing CRS object, or 3) a shortcut reference string to a predefined projection; run \code{names(fm_wkt_predef())} for valid predefined projections.
  \item \texttt{doCheckCRSArgs} default \texttt{TRUE}, must be set to \texttt{FALSE} by package developers including CRS in an \S4 class definition to avoid uncontrollable loading of the \pkg{rgdal} namespace.
  \item \texttt{args} An optional list of name/value pairs to add to and/or override the PROJ4 arguments in \texttt{projargs}. \texttt{name=value} is converted to "+name=value", and \texttt{name=NA} is converted to "+name".
  \item \texttt{SRS_string} a WKT2 string defining the coordinate system; see \code{sp::CRS}. This takes precedence over \texttt{projargs}.
\end{itemize}

Details

The first two elements of the \texttt{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: \texttt{oblique=c(0, 45)}

Polar: \texttt{oblique=c(0, 90)}

Quasi-transversal: \texttt{oblique=c(0, 0, 90)}

Satellite orbit viewpoint: \texttt{oblique=c(lon0-time*v1, orbitangle, orbit0+time*v2)}, where \texttt{lon0} is the longitude at which a satellite orbit crosses the equator at \texttt{time=0}, when the satellite is at an angle \texttt{orbit0} further along in its orbit. The orbital angle relative to the equatorial plane is \texttt{orbitangle}, and \texttt{v1} and \texttt{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 \texttt{oblique[2]} or \texttt{oblique[3]} are non-zero, the resulting projection is only correct for perfect spheres.
Value

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

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

\begin{verbatim}
    crs
\end{verbatim}

The basic \texttt{sp::CRS} object

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

\texttt{sp::CRS()}, \texttt{fm_crs_wkt}, \texttt{fm_sp_get_crs()}, \texttt{fm_identical_CRS()}

Examples

\begin{verbatim}
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 RPROJ6 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")
  }
}
\end{verbatim}

\begin{verbatim}

\end{verbatim}

\texttt{fm_crs}

\textit{Obtain coordinate reference system object}

Description

Obtain an \texttt{sf::crs} or \texttt{fm_crs} object from a spatial object, or convert crs information to construct a new \texttt{sf::crs} object.
Usage

fm_crs(x, ..., crs-only = FALSE)

fm_crs_oblique(x)

fm_crs_oblique(x) <- value

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

## Default S3 method:
fm_crs(x, ..., crsonly = FALSE)

st_crs.fm_crs(x, ...)

## S3 method for class 'fm_crs'
x$name

## S3 method for class 'fm_crs'
fm_crs(x, ..., crs-only = FALSE)

## S3 method for class 'inla.CRS'
fm_crs(x, ..., crs-only = FALSE)

## S3 method for class 'character'
fm_crs(x, ..., crs-only = FALSE)

## S3 method for class 'Spatial'
fm_crs(x, ..., crs-only = FALSE)

## S3 method for class 'SpatVector'
fm_crs(x, ..., crs-only = FALSE)

## S3 method for class 'SpatRaster'
fm_crs(x, ..., crs-only = FALSE)

## S3 method for class 'sf'
fm_crs(x, ..., crs-only = FALSE)

## S3 method for class 'sfc'
fm_crs(x, ..., crs-only = FALSE)

## S3 method for class 'sfg'
fm_crs(x, ..., crs-only = FALSE)

## S3 method for class 'inla.mesh'
fm_crs(x, ..., crs-only = FALSE)
# S3 method for class 'inla.mesh.lattice'
fm_crs(x, ..., crsonly = FALSE)

# S3 method for class 'inla.mesh.segment'
fm_crs(x, ..., crsonly = FALSE)

fm_wkt_preadef()

**Arguments**

- **x**: Object to convert to crs or to extract crs information from. If character, a string suitable for sf::st_crs(x), or the name of a predefined wkt string from 'names(fm_wkt_preadef())'.
- **...**: Additional parameters. Not currently in use.
- **crsonly**: logical; if TRUE, remove any oblique information for fm_crs class objects and return a pure crs class object. Default: FALSE.
- **value**: 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.
- **name**: element name

**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 observer's 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 sf::crs object or an fm_crs object, depending on if the coordinate reference system described by the parameters can be expressed with a pure crs object or not.

A crs object (sf::st_crs()) or a fm_crs object. An S3 fm_crs object is a list with elements crs and oblique.

fm_wkt_preadef returns a WKT2 string defining a projection
Methods (by generic)

• $: For a fm_crs object x, x$name calls the accessor method for the crs object inside it. If name is "crs", the internal crs object itself is returned. If name is "oblique", the internal oblique angle parameter vector is returned.

Functions

• st_crs.fm_crs(): st_crs(x, ...) is equivalent to fm_crs(x, ..., crsonly = TRUE) when x is a fm_crs object.

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

sf::st_crs(), fm_crs_wkt

Examples

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")
## Not run:
names(fm_wkt_predef())

## End(Not run)

--

fm_evaluate (61)

Methods for projecting to/from an inla.mesh

Description

Calculate a lattice projection to/from an inla.mesh

Usage

fm_evaluate(...)

## S3 method for class 'inla.mesh'
fm_evaluate(mesh, field, ...)

## S3 method for class 'inla.mesh.1d'
fm_evaluate(mesh, field, ...)
## fm_evaluate

### S3 method for class 'fm_evaluator'

```r
fm_evaluate(projector, field, ...)
```

```r
fm_evaluator(...)```

```r
fm_evaluator_inla_mesh(mesh, loc = NULL, crs = NULL, ...)
```

```r
fm_evaluator_inla_mesh_1d(mesh, loc, ...)
```

```r
fm_evaluator_lattice(
  mesh,
  xlim = NULL,
  ylim = NULL,
  dims = c(100, 100),
  projection = NULL,
  crs = NULL,
  ...
)
```

### S3 method for class 'inla.mesh'

```r
fm_evaluator(mesh, loc = NULL, lattice = NULL, crs = NULL, ...)
```

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

```r
fm_evaluator(mesh, loc = NULL, xlim = mesh$interval, dims = 100, ...)
```

## Arguments

- `...`: Additional arguments passed on to methods.
- `mesh`: An `inla.mesh` or `inla.mesh.1d` object.
- `field`: Basis function weights, one per mesh basis function, describing the function to be evaluated at the projection locations.
- `projector`: An `fm_evaluator` object.
- `loc`: Projection locations. Can be a matrix, `SpatialPoints`, `SpatialPointsDataFrame`, `sf`, `sfc`, or `sfg` object.
- `crs`: An optional CRS or `inla.CRS` object associated with `loc` and/or `lattice`.
- `xlim`: X-axis limits for a lattice. For R2 meshes, defaults to covering the domain.
- `ylim`: Y-axis limits for a lattice. For R2 meshes, defaults to covering the domain.
- `dims`: Lattice dimensions.
- `projection`: One of `c("default", "longlat", "longsinlat", "mollweide")`.
- `lattice`: An `inla.mesh.lattice()` object.

## Value

For `fm_evaluate(mesh, ...)`, a list with projection information. For `fm_evaluator(mesh, ...)`, an `fm_evaluator` object. For `fm_evaluate(projector, field, ...)`, a field projected from the mesh onto the locations given by the projector object.
### 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)`

---

**Functions**

- `fm_evaluator(inla.mesh)`: The arguments are passed on to `fm_evaluator_lattice()` if no loc or lattice is provided.

**Author(s)**

Finn Lindgren <finn.lindgren@gmail.com>

**See Also**

`inla.mesh, inla.mesh.1d, inla.mesh.lattice`

**Examples**

```r
if (bru_safe_inla()) {
  n <- 20
  loc <- matrix(runif(n * 2), n, 2)
  mesh <- INLA::inla.mesh.create(loc, refine = list(max.edge = 0.05))
  proj <- fm_evaluator(mesh)
  field <- cos(mesh$loc[, 1] * 2 * pi * 3) * sin(mesh$loc[, 2] * 2 * pi * 7)
  image(proj$x, proj$y, fm_evaluate(proj, field))
}

if (bru_safe_inla() && require(rgl)) {
  plot(mesh, rgl = TRUE, col = field, draw.edges = FALSE, draw.vertices = FALSE)
}
```
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**

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

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 its 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

**Extract CRS information**

**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. For newer code, use `fm_crs()` instead, that returns crs objects, and use `fm_as_sp_crs()` to convert to old style sp::CRS objects.
Value

A CRS object, or NULL if no valid CRS identified

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

Examples

```r
## 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_transform**  
*Object coordinate transformation*

Description

Handle transformation of various inla objects according to coordinate reference systems of crs (from sf::st_crs()), fm_crs, sp::CRS or INLA::inla.CRS class.

Usage

```r
fm_transform(x, crs = fm_crs(x), ...)
```

```
## Default S3 method:
fm_transform(x, crs = fm_crs(x), ..., crs0 = NULL)
```

```
## S3 method for class 'matrix'
fm_transform(x, crs = NULL, ..., passthrough = FALSE, crs0 = NULL)
```

```
## S3 method for class 'list'
fm_transform(x, crs = fm_crs(x), ...)
```

```
## S3 method for class 'sf'
fm_transform(x, crs = fm_crs(x), ..., passthrough = FALSE)
```

```
## S3 method for class 'sfc'
fm_transform(x, crs = fm_crs(x), ..., passthrough = FALSE)
```

```
## S3 method for class 'sfg'
fm_transform(x, crs = fm_crs(x), ..., passthrough = FALSE)
```
### fm_wkt_as_wkt_tree

**Description**

Conversion between WKT and a tree representation

**Usage**

```r
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
...
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

---

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_ellipsoid_radius(x)

## Default S3 method:
fm_ellipsoid_radius(x)

## S3 method for class 'character'
fm_ellipsoid_radius(x)

fm_wkt_set_ellipsoid_radius(wkt, radius)
fm_ellipsoid_radius(x) <- value

## S3 replacement method for class 'character'
fm_ellipsoid_radius(x) <- value

## S3 replacement method for class 'CRS'
fm_ellipsoid_radius(x) <- value

## S3 replacement method for class 'inla.CRS'
fm_ellipsoid_radius(x) <- value

## S3 replacement method for class 'crs'
fm_ellipsoid_radius(x) <- value

## S3 replacement method for class 'fm_crs'
fm_ellipsoid_radius(x) <- value

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_lengthunit(crs)

fm_crs_set_lengthunit(crs, unit)

fm_length_unit(x)

## Default S3 method:
fm_length_unit(x)

## S3 method for class 'character'
fm_length_unit(x)

fm_length_unit(x) <- value

## S3 replacement method for class 'character'
fm_length_unit(x) <- value

## S3 replacement method for class 'CRS'
fm_length_unit(x) <- value

## S3 replacement method for class 'inla.CRS'
fm_length_unit(x) <- value

## S3 replacement method for class 'crs'
fm_length_unit(x) <- value

## S3 replacement method for class 'fm_crs'
fm_length_unit(x) <- value

fm_wkt(crs)

fm_crs_get_wkt(crs)
Arguments

- **wkt** A WKT2 character string
- **crs** A sp::CRS or inla.CRS object
- **x** crs object to extract value from or assign values in
- **radius** numeric; The new radius value
- **value** Value to assign
- **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_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_length_unit<-, a crs object with altered length units. Note that the length unit for the ellipsoid radius is unchanged.

For fm_wkt, WKT2 string.

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

- fm_sp_get_crs()

Examples

```r
## 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 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 bru().
- **...**: additional, unused arguments.
- **data**: A data.frame or SpatialPointsDataFrame of covariates needed for sampling.
- **formula**: A formula where the right hand side defines an R expression to evaluate for each generated sample. If NULL, the latent and hyperparameter states are returned as named list elements. See Details for more information.
- **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 INLA::inla.posterior.sample
- **num.threads**: Specification of desired number of threads for parallel computations. Default NULL, leaves it up to INLA. When seed != 0, overridden to "1:1"
- **include**: Character vector of component labels that are needed by the predictor expression; Default: NULL (include all components that are not explicitly excluded)
exclude

Character vector of component labels that are not used by the predictor expression. The exclusion list is applied to the list as determined by the include parameter; Default: NULL (do not remove any components from the inclusion list).

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

```r
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 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))
  }
}
gg

**ggplot2 geomes for inlabru related objects**

**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**

```r
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**

```r
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$boundary) +
  gg(gorillas$nest)
}
```
Description

This geom constructor will simply call `gg.prediction` for the data provided.

Usage

```r
## 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

```r
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 simultaneously 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 useful 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
gg.inla.mesh

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

---

**gg.inla.mesh**  
*Geom for inla.mesh objects*

**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**

```r
## 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**

- `data`  
  An INLA::inla.mesh object.

- `color`  
  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 `colour` is also recognised.

- `alpha`  
  A vector of scalar values setting the alpha value of the colors provided.

- `edge.color`  
  Color of the mesh edges.

- `interior`  
  If TRUE, plot the interior boundaries of the mesh.

- `int.color`  
  Color used to plot the interior boundaries.

- `exterior`  
  If TRUE, plot the exterior boundaries of the mesh.
ext.color  Color used to plot the interior boundaries.
crs       A CRS object defining the coordinate system to project the mesh to before plotting.
mask      A SpatialPolygon defining the region that is plotted.
nx        Number of pixels in x direction (when plotting using the color parameter).
ny        Number of pixels in y direction (when plotting using the color parameter).
...       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

```r
if (bru_safe_inla() &&
    require(ggplot2, quietly = TRUE) &&
    require(ggpolyrep, 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() +
  ggplot() +
```
gg.inla.mesh.1d

```r
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**

```r
## 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.
gg.prediction

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  Geom for predictions

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

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

Arguments
data  A prediction object, usually the result of a predict.bru() call.
mapping  a set of aesthetic mappings created by aes. These are passed on to geom_line.
ribbon  If TRUE, plot a ribbon around the line based on the smallest and largest quantiles present in the data, found by matching names starting with q and followed by a numerical value. inla()-style numeric+"quant" names are converted to inlabru style before matching.
alpha  The ribbons numeric alpha (transparency) level in [0,1].
bar  If TRUE plot boxplot-style summary for each variable.
...  Arguments passed on to geom_line.

Value
Concatenation of a geom_line value and optionally a geom_ribbon value.
gg.prediction

See Also

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

Examples

```r
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 simultaneously 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 useful 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
# `ribbon` parameter

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

# Much like the other geometries 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**

```r
## S3 method for class 'RasterLayer'

gg(
  data,
  mapping = ggplot2::aes(x = .data[["x"]], y = .data[["y"]], fill = .data[["layer"]]),
  ...
)
```

```r
```
gg.SpatialGridDataFrame

Description

Coerces input SpatialGridDataFrame to SpatialPixelsDataFrame and calls gg.SpatialPixelsDataFrame() to plot it. Requires the ggplot2 package.

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()
## Usage

```r
## 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 geoms for spatial data: `gg.SpatialLines()`, `gg.SpatialPixelsDataFrame()`, `gg.SpatialPixels()`, `gg.SpatialPoints()`, `gg.SpatialPolygons()`, `gg()`, `gm()`

### Examples

```r
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$boundary) +
    gg(gorillas$nests)

  # Load pantropical dolphin data
  data("mexdolphin", package = "inlabru")

  # Plot the pantropical 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))
}

---

**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**

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

**Arguments**

- `data` A `SpatialLines` or `SpatialLinesDataFrame` object.
- `mapping` Aesthetic mappings created by `ggplot2::aes` or `ggplot2::aes_` used to update the default mapping. The default mapping is `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])]])`.
- `crs` A CRS object defining the coordinate system to project the data to before plotting.
- `...` Arguments passed on to `ggplot2::geom_segment`.

**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

```r
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$boundary) +
    gg(gorillas$nests)

  # Load pantropical dolphin data
  data("mexdolphin", package = "inlabru")

  # Plot the pantropical 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))
}

---

**gg.SpatialPixels**  
*Geom for SpatialPixels objects*

**Description**

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

**Usage**

```r
## 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**

```r
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)
}
```
Description

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

Usage

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

Arguments

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

Value

A `geom_tile` return value.

See Also

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

Examples

```r
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$boundary) +
  gg(gorillas$nests)
#
# Load pantropical dolphin data

data("mexdolphin", package = "inlabru")
#
# Plot the pantropical 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))

---

**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.
## S3 method for class `SpatialPoints`

```r
gg(data, mapping = NULL, crs = NULL, ...)
```

### Arguments

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

### 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

```r
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$boundary) +
    gg(gorillas$ests)

  # Load pantropical dolphin data
  data("mexdolphin", package = "inlabru")

  # Plot the pantropical survey boundary, ship transects and dolphin sightings
  ggplot() +
    gg(mexdolphin$ship$trans) +
    gg(mexdolphin$boundary) +
    gg(mexdolphin$dolphin)
}
```
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))

---

**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**

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

**Arguments**

- **data**
  A `SpatialPolygons` or `SpatialPolygonsDataFrame` object.

- **mapping**
  Aesthetic mappings created by `aes` used to update the default mapping. The default mapping is `ggplot2::aes(x = long, y = lat, group = group)`.

- **crs**
  A CRS object defining the coordinate system to project the data to before plotting.

- **...**
  Arguments passed on to `geom_polyline`. Unless specified by the user, the arguments `colour = "black"` (polygon colour) and `alpha = 0.2` (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**


**Examples**

```r
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$boundary) +
  gg(gorillas$nests)
  # Load pantropical dolphin data
  data("mexdolphin", package = "inlabru")
  # Plot the pantropical 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

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

# Visualize data annotations: point size by dolphin group size

```r
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

```r
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: \texttt{glplot.SpatialLines()}, \texttt{glplot.SpatialPoints()}, \texttt{glplot.inla.mesh()}, \texttt{glplot()}

Examples

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

  # Load pantropoical dolphin data
  data("mexdolphin", package = "inlabru")

  # Show the globe
  globe()

  # Add mesh, ship transects and dolphin sightings stored
  # as inla.mesh, SpatialLines and SpatialPoints objects, respectively
  glplot(mexdolphin$mesh)
  glplot(mexdolphin$samplers)
  glplot(mexdolphin$points)
}
## End(Not run)
```

---

glplot  

\textit{Render Spatial* and inla.mesh objects using RGL}

Description

\texttt{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

\texttt{glplot(object, \ldots)}
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("mexdolphin", package = "inlabru")

  # Show the globe
  globe()

  # Add mesh, ship transects and dolphin sightings stored
  # as inla.mesh, SpatialLines and SpatialPoints objects, respectively
  glplot(mexdolphin$mesh)
  glplot(mexdolphin$samplers)
  glplot(mexdolphin$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, ...)

---

glplot.inla.mesh
Arguments

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

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("mexdolphin", package = "inlabru")
  # Show the globe
globe()

  # Add mesh, ship transects and dolphin sightings stored
  # as inla.mesh, SpatialLines and SpatialPoints objects, respectively
glplot(mexdolphin$mesh)
glplot(mexdolphin$samplers)
glplot(mexdolphin$points)
}
## End(Not run)
### Arguments

- **object**: a SpatialLines or SpatialLinesDataFrame object.
- **add**: If TRUE, add the lines to an existing plot. If FALSE, create new plot.
- **...**: Parameters passed on to rgl.linestrips().

### See Also

Other inlabru RGL tools: `globe()`, `glplot.SpatialPoints()`, `glplot.inla.mesh()`, `glplot()`

### Examples

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

  # Load pantropoical dolphin data
  data("mexdolphin", package = "inlabru")

  # Show the globe
  globe()

  # Add mesh, ship transects and dolphin sightings stored
  # as inla.mesh, SpatialLines and SpatialPoints objects, respectively
  glplot(mexdolphin$mesh)
  glplot(mexdolphin$samplers)
  glplot(mexdolphin$points)

}## End(Not run)
```

### Description

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

### Usage

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

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

See Also

Other inlabru RGL tools: `globe()`, `glplot.SpatialLines()`, `glplot.inla.mesh()`, `glplot()`

Examples

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

  # Load pantropeic dolphin data
  data("mexdolphin", package = "inlabru")

  # Show the globe
  globe()

  # Add mesh, ship transects and dolphin sightings stored
  # as inla.mesh, SpatialLines and SpatialPoints objects, respectively
  glplot(mexdolphin$mesh)
  glplot(mexdolphin$samplers)
  glplot(mexdolphin$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, ...)"
gmap

Plot a map using extent 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 ggeom() 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 centred 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$nests, maptype = "satellite") +
    gm(gorillas$boundary) +
    gm(gorillas$nests, color = "white", size = 0.5)
  }
}

## End(Not run)
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.
  - **slopalngle**: 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


Examples

```r
if (bru_safe_inla() &&
    require(ggplot2, quietly = TRUE) &&
    require(ggpolypath, quietly = TRUE)) {
    data(gorillas, package = "inlabru") # get the data

    # 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$plotsample$nests)
}
```

inla.stack.mjoin  
Join stacks intended to be run with different likelihoods

Description

Join stacks intended to be run with different likelihoods

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>List of stacks that contain vector observations (existing multi-likelihood observation matrices are also permitted)</td>
</tr>
<tr>
<td>compress</td>
<td>If TRUE, compress the model by removing duplicated rows of effects, replacing the corresponding A-matrix columns with a single column containing the sum.</td>
</tr>
<tr>
<td>remove.unused</td>
<td>If TRUE, compress the model by removing rows of effects corresponding to all-zero columns in the A matrix (and removing those columns).</td>
</tr>
<tr>
<td>old.names</td>
<td>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 &quot;BRU.response&quot;)</td>
</tr>
</tbody>
</table>
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, mexdolphin, 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 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 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 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**

```r
# 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"))
}
```

---

**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 points 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 SpatialLinesDataFrame 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 \texttt{int.args$method="stable"}
(default) or "direct", where the latter uses the integration points directly, without aggregating to
the mesh vertices.

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

\textbf{Usage}

\begin{verbatim}
ipoints(
  samplers = NULL,
  domain = NULL,
  name = NULL,
  group = NULL,
  int.args = NULL,
  project = NULL
)
\end{verbatim}

\textbf{Arguments}

\begin{description}
  \item[samplers] 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 
\texttt{SpatialPolygon} or a \texttt{SpatialLinesDataFrame} with a weight column defining
the width of the a transect line, and optionally further columns used by the 
\texttt{group} argument, or NULL. When \texttt{domain} is NULL, \texttt{samplers} may also be an
\texttt{inla.mesh.1d} or \texttt{inla.mesh} object, that is then treated as a domain argument
instead.
  \item[domain] Either
    \begin{itemize}
      \item when \texttt{samplers} is a 1D interval(s) definition only, \texttt{domain} can be a single
        integer for the number of integration points to place in each 1D interval, 
        overriding \texttt{int.args$["nsub1"]}, and otherwise
      \item when \texttt{samplers} is NULL, \texttt{domain} can be a numeric vector of points, each
        given integration weight 1 (and no additional points are added in between),
      \item an \texttt{inla.mesh.1d} object for continuous 1D integration, or
      \item an \texttt{inla.mesh.2d} object for continuous 2D integration.
    \end{itemize}
  \item[name] Character array stating the name of the domains dimension(s). If NULL, the
    names are taken from coordinate names from \texttt{samplers} for \texttt{Spatial}* objects,
    otherwise "x", "y", "z" for 2D regions and "x" for 1D regions
  \item[group] Column names of the \texttt{samplers} object (if applicable) for which the integration
    points are calculated independently and not merged when aggregating to mesh
    nodes.
  \item[int.args] List of arguments passed to \texttt{bru_int_polygon}.
    \begin{itemize}
      \item method: "stable" (to aggregate integration weights onto mesh nodes) or
        "direct" (to construct a within triangle/segment integration scheme without
        aggregating onto mesh nodes)
    \end{itemize}
\end{description}
ipoints

- 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

```r
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**

```r
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 SpatialPoints(DataFrame) object.
- **samplers**: A data frame or Spatial[Points/Lines/Polygons]DataFrame objects.
- **domain**: Named list of domain definitions.
- **ips**: Integration points (overrides samplers).
- **formula**: If NULL, 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 `like()`.
- **options**: See `bru_options_set()`.

### Value

An `bru()` object.

### Examples

```r
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$mesh),
    options = list(control.inla = list(int.strategy = "eb")))
```
like <- predict(fit, pixels(gorillas$mesh), ~ exp(mySmooth + Intercept))

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

like

Description

Likelihood construction for usage with bru()

Usage

like(
  formula = . ~ .,
  family = "gaussian",
  data = NULL,
  response_data = NULL,
  mesh = NULL,
  E = NULL,
  Ntrials = NULL,
  weights = 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()$likelihood)), inlabru supports fitting latent Gaussian Cox processes via family = "cp". As an alternative to bru(), the lgcp() function provides a convenient interface to fitting Cox processes.
- **data**: Likelihood-specific data, as a data.frame or SpatialPointsDataFrame object.
- **response_data**: Likelihood-specific data for models that need different size/format for inputs and response variables, as a data.frame or SpatialPointsDataFrame object.
- **mesh**: An inla.mesh object. Obsolete.
- **E**: Exposure parameter for family = 'poisson' passed on to INLA::inla. Special case if family is 'cp': rescale all integration weights by E. Default taken from options$E, normally 1.
- **Ntrials**: A vector containing the number of trials for the 'binomial' likelihood. Default taken from options$Ntrials, normally 1.
- **weights**: Fixed (optional) weights parameters of the likelihood, so the log-likelihood[i] is changed into weights[i] * log_likelihood[i]. Default value is 1. WARNING: The normalizing constant for the likelihood is NOT recomputed, so ALL marginals (and the marginal likelihood) must be interpreted with great care.
- **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

control.family A optional list of INLA::control.family options

options A bru_options options object or a list of options passed on to `bru_options()`

.envir The evaluation environment to use for special arguments (E, Ntrials, and weights) if not found in `response_data` or data. Defaults to the calling environment.

... For `like_list.bru_like`, one or more `bru_like` objects

object A list of `bru_like` objects

envir An optional environment for the new `bru_like_list` object

x bru_like_list object from which to extract element(s)

i 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

```r
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)

mexdolphin

Pan-tropical spotted dolphins in the Gulf of Mexico
Description

This 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(mexdolphin)

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


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

```r
if (bru_safe_inla(quietly = TRUE) && require("ggplot2", quietly = TRUE) && require("ggpolypath", quietly = TRUE)) {
  data(mexdolphin, package = "inlabru")
  ggplot() +
```
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.
covar: An SpatialPointsDataFrame containing sea depth estimates.
multiplot

Source
Library MRSea.

References
NONE YET

Examples
if (bru_safe_inla() &&
    require(ggplot2, quietly = TRUE) &&
    require(ggpolypath, 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 matrix(c(1,2,3,3), nrow=2, byrow=TRUE), 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)/
Examples

```r
if (require("ggplot2", quietly = TRUE)) {
  df <- data.frame(x = 1:10, y = 1:10, z = 11:20)
  pl1 <- ggplot(data = df) +
    geom_line(mapping = aes(x, y), color = "red")
  pl2 <- ggplot(data = df) +
    geom_line(mapping = aes(x, z), color = "blue")
  multiplot(pl1, pl2, cols = 2)
}
```

Description

Generate SpatialPixels covering an inla.mesh

Usage

```r
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

```r
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

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

Arguments

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

Value

an object of class gg

Examples

```r
## 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

Plot prediction using ggplot2

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:

```r
ggplot() +
  gg(xpost)
```

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

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

```r
xipost
```

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

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

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

```r
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:

```r
ggplot() +
  gg(xipost)
```

# For ease of use we can also type

```r
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 of using the
# \code{ribbon} parameter

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

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

```r
ggplot() +
  gg(xipost) +
  ```


```r
plotsample <- function(xipost, mapping = aes(y = median), ribbon = FALSE, color = "red")
{
  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 `SpatialPointsDataFrame` defining the points that are to be sampled by the plot sample.
- `boundary`: A `SpatialPolygonsDataFrame` 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

```r
# Some features require the raster package
if (require("raster", quietly = TRUE) && require("ggplot2", quietly = TRUE)) {
  data(gorillas, package = "inlabru")
  plotpts <- plotsample(gorillas$nests, gorillas$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**

```r
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$n.

**Examples**

```r
# Some features require the raster package
if (require("raster", quietly = TRUE) && require("ggplot2", quietly = TRUE)) {
  data(gorillas, package = "inlabru")
  plotpts <- plotsample(gorillas$nests, gorillas$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)
}
```
Poison1_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

```r
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

```r
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:

<table>
<thead>
<tr>
<th>x</th>
<th>count</th>
<th>exposure</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Examples

```r
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
  )
```
```
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**

```r
## S3 method for class 'bru'
predict(
  object,
  data = NULL,
  formula = NULL,
  n.samples = 100,
  seed = 0L,
  probs = c(0.025, 0.5, 0.975),
  num.threads = NULL,
  include = NULL,
  exclude = NULL,
  drop = FALSE,
  ...
)
```

Arguments

- **object**: An object obtained by calling `bru()` or `lgcp()`.
- **data**: A data.frame or `SpatialPointsDataFrame` 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 `NULL`, 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 `inla.posterior.sample`
- **probs**: A numeric vector of probabilities with values in \([0, 1]\), passed to `stats::quantile`
- **num.threads**: Specification of desired number of threads for parallel computations. Default `NULL`, leaves it up to INLA. When `seed != 0`, overridden to "1:1"
- **include**: Character vector of component labels that are needed by the predictor expression; Default: `NULL` (include all components that are not explicitly excluded)
- **exclude**: Character vector of component labels that are not used by the predictor expression. The exclusion list is applied to the list as determined by the `include` parameter; Default: `NULL` (do not remove any components from the inclusion list)
- **drop**: logical; If `keep = FALSE`, data is a `Spatial*DataFrame`, and the prediction summary has the same number of rows as data, then the output is a `Spatial*DataFrame` object. Default `FALSE`.
- **...**: Additional arguments passed on to `inla.posterior.sample`

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

```r
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.inla.mesh(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 \code{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 SpatialPixelsDataFrame with the columns we are looking for

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

This is the robins_subset dataset, which is a subset of the full robins data set used to demonstrate a spatially varying trend coefficient model in Meehan et al. 2019. The dataset includes American Robin counts, along with time, location, and effort information, from Audubon Christmas Bird Counts (CBC) conducted in six US states between 1987 and 2016.

Usage

data(robins_subset)

Format

The data are a data.frame with variables

circle: Four-letter code of the CBC circle.
bcr: Numeric code for the bird conservation region encompassing the count circle.
state: US state encompassing the count circle.
year: calendar year the count was conducted.
std_yr: transformed year, with 2016 = 0.
count: number of robins recorded.
log_hrs: the natural log of party hours.
lon: longitude of the count circle centroid.
lat: latitude of the count circle centroid.
obs: unique record identifier.

Source

https://github.com/tmeeha/inlaSVCBC

References

Examples

```r
if (require(ggplot2, quietly = TRUE)) {
  data(robins_subset, package = "inlabru")  # get the data
  # plot the counts for one year of data
  ggplot(robins_subset[robins_subset$std_yr == 0, ]) +
    geom_point(aes(lon, lat, colour = count + 1)) +
    scale_colour_gradient(low = "blue", high = "red", trans = "log")
}
```

---

**row_kron**

**Row-wise Kronecker products**

**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**

```r
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 INLA::inla.spde.make.A.
- `n.repl`: The maximum replicate index, in the sense used in INLA::inla.spde.make.A().
- `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 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

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

Arguments

- `mesh`: An INLA::inla.mesh object
- `loglambda`: vector or matrix; A vector of log intensities at the mesh vertices (for higher order basis functions, e.g. for inla.mesh.1d meshes, loglambda should be given as mesh$m basis function weights rather than the values at the mesh$n 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 SpatialPolygonsDataFrame or inla.mesh 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) SpatialPointsDataFrame (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

```r
# 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 = "|")
}
```

```r
# 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
```

seals

```
}
  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**


**Examples**

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


Examples

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

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

Value

SpatialLinesDataFrame

Examples

# Create a data frame defining three lines
lns <- 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(lns,
  start.cols = c("xs", "ys"),
  end.cols = c("xe", "ye")
)
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$nests, gorillas$boundary)
  class(gp)

  # Plot it
  plot(gp)
}
spde.posterior

Posterior distribution of SPDE hyper parameters and Matern correlation or covariance function.

Description

Calculate posterior distribution of the range, log(range), variance, or log(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 inla.
name Character stating the name of the SPDE effect, see names(result$summary.random).
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

Convert a data.frame of boundary points into a SpatialPolygonsDataFrame

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 ba 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.
**stransform**

**Description**

These functions still attempt to do their job, but will be removed in a future version.

**Usage**

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

**Arguments**

- **data**: A data.frame of points describing the boundary of the polygon
- **cols**: Column names of the x and y coordinates within the data
- **crs**: Coordinate reference system of the points
- **to.crs**: Coordinate reference system for the SpatialLines output.

**Value**

SpatialPolygonsDataFrame

**Examples**

```r
# 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) && require(ggpolypath, quietly = TRUE)) {
  # Plot it!
  ggplot() +
  gg(pol)
}
```
Usage

stransform(splist, crs)
init.tutorial()
ibm_valid_input(...)

## S3 method for class 'bru_mapper_inla_mesh_2d'
ibm_amatrix(...)

## S3 method for class 'bru_mapper_inla_mesh_1d'
ibm_amatrix(...)

## S3 method for class 'bru_mapper_index'
ibm_amatrix(...)

## S3 method for class 'bru_mapper_linear'
ibm_amatrix(...)

## S3 method for class 'bru_mapper_matrix'
ibm_amatrix(...)

## S3 method for class 'bru_mapper_factor'
ibm_amatrix(...)

bru_mapper_offset(...)

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

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

## S3 method for class 'bru_mapper_offset'
ibm_amatrix(...)

## S3 method for class 'bru_mapper_multi'
ibm_amatrix(...)

## S3 method for class 'bru_mapper_collect'
ibm_amatrix(...)

eval_SpatialDF(...)

Arguments

splist  list of Spatial* objects
crs     Coordinate reference system to change to
... Usually passed on to other methods

Value

stransform

Functions

- **stransform()**: Coordinate transformation for spatial objects
  
  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. Returns a list of Spatial* objects.
  
  Deprecated in favour of the `fm_transform` methods.

- **init.tutorial()**: Global setting for tutorial sessions.
  
  Use `bru_options_set()` to set specific options instead instead. In versions <= 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".

- **ibm_valid_input()**: Use case changed to `ibm_invalid_output`

- **ibm_amatrix(bru_mapper_inla_mesh_2d)**: Replaced by `ibm_jacobian`

- **ibm_amatrix(bru_mapper_inla_mesh_1d)**: Replaced by `ibm_jacobian`

- **ibm_amatrix(bru_mapper_index)**: Replaced by `ibm_jacobian`

- **ibm_amatrix(bru_mapper_linear)**: Replaced by `ibm_jacobian`

- **ibm_amatrix(bru_mapper_matrix)**: Replaced by `ibm_jacobian`

- **ibm_amatrix(bru_mapper_factor)**: Replaced by `ibm_jacobian`

- **bru_mapper_offset()**: Creates a `bru_mapper_const()` mapper.

- **ibm_n(bru_mapper_offset)**: Replaced by `bru_mapper_const` methods

- **ibm_values(bru_mapper_offset)**: Replaced by `bru_mapper_const` methods

- **ibm_amatrix(bru_mapper_offset)**: Replaced by `bru_mapper_const` methods

- **ibm_amatrix(bru_mapper_multi)**: Replaced by `ibm_jacobian`

- **ibm_amatrix(bru_mapper_collect)**: Replaced by `ibm_jacobian`

- **eval_SpatialDF()**: Replaced by the generic `eval_spatial`

**Author(s)**

Finn Lindgren <finn.lindgren@gmail.com>

Fabian E. Bachl <bachlfab@gmail.com>
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)) {
  multplot(
    ggplot() +
    gg(gorillas$mesh) +
    ggtitle("Original mesh"),
    ggplot() +
    gg(tmesh) +
    ggtitle("Transformed mesh")
  )
}

## 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)

---

st_check_dim  

Check for "XYZ", "XYM" and "XYZM" sfg classes

Description

Check for "XYZ", "XYM" and "XYZM" sfg classes

Usage

st_check_dim(sfc)

Arguments

sfc  An sfc object
**st_check_polygon**

**Value**

LOGICAL indicating if any sfg element of the sfc object has class "XYZ", "XYM" or "XYZM". Internal function used to check for 3 and 4 dimensional objects.

**Description**

It seems as though st_polygon does not check this. For now only implements a basic check for disjoint regions using st_within()

**Usage**

```r
st_check_polygon(sfg)
```

**Arguments**

- `sfg` A POLYGON sfg object

**Value**

LOGICAL; TRUE if the sfg holes are entirely inside the outer ring, and are disjoint, otherwise FALSE. When FALSE, the attribute Message is set to a character vector describing the detected reasons.

---

**st_signed_area**

**Calculate signed area for polygon**

**Description**

Calculate signed area for polygon

**Usage**

```r
st_signed_area(sfg)
```

**Arguments**

- `sfg` A POLYGON sfg object

**Value**

Returns the signed area. Negative values indicate anti-clockwise winding direction.
Summary for an inlabru fit

Description

Takes a fitted bru object produced by bru() or lgcp() and creates various summaries from it.

Usage

```r
## S3 method for class 'bru'
summary(object, verbose = FALSE, ...)

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

Arguments

- `object`: An object obtained from a bru() or lgcp() call
- `verbose`: logical; If TRUE, include more details of the component definitions. If FALSE, only show basic component definition information. Default: FALSE
- `...`: arguments passed on to component summary functions, see summary.component()
- `x`: An summary_bru2 object

Examples

```r
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:
```
summary.bru_info

Methods for bru_info objects

Description

Methods for bru_info objects

Usage

## S3 method for class 'bru_info'
summary(object, verbose = TRUE, ...)

## 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Object to operate on</td>
</tr>
<tr>
<td>verbose</td>
<td>logical; If TRUE, include more details of the component definitions. If FALSE, only show basic component definition information. Default: TRUE</td>
</tr>
<tr>
<td>...</td>
<td>Arguments passed on to other methods</td>
</tr>
<tr>
<td>x</td>
<td>A summary_bru_info object to be printed</td>
</tr>
<tr>
<td>method</td>
<td>character; The type of estimation method used</td>
</tr>
<tr>
<td>inlabru_version</td>
<td>character; inlabru package version. Default: NULL, for automatically detecting the version</td>
</tr>
<tr>
<td>INLA_version</td>
<td>character; INLA package version. Default: NULL, for automatically detecting the version</td>
</tr>
</tbody>
</table>

Description
mapper object summaries

Usage

```r
## S3 method for class 'bru_mapper'
summary(object, ..., prefix = "", initial = prefix, depth = 1)

## S3 method for class 'bru_mapper_multi'
summary(object, ..., prefix = "", initial = prefix, depth = 1)

## S3 method for class 'bru_mapper_pipe'
summary(object, ..., prefix = "", initial = prefix, depth = 1)

## S3 method for class 'bru_mapper_collect'
summary(object, ..., prefix = "", initial = prefix, depth = 1)

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>bru_mapper object to summarise</td>
</tr>
<tr>
<td>...</td>
<td>Unused arguments</td>
</tr>
<tr>
<td>prefix</td>
<td>character prefix for each line. Default &quot;&quot;.</td>
</tr>
<tr>
<td>initial</td>
<td>character prefix for the first line. Default initial=prefix.</td>
</tr>
<tr>
<td>depth</td>
<td>The recursion depth for multi/collection/pipe mappers. Default 1, to only show the collection, and not the contents of the sub-mappers.</td>
</tr>
<tr>
<td>x</td>
<td>Object to be printed</td>
</tr>
</tbody>
</table>

summary.bru_mapper  mapper object summaries

## S3 method for class 'bru_mapper'
summary(object, ..., prefix = "", initial = prefix, depth = 1)

## S3 method for class 'bru_mapper_multi'
summary(object, ..., prefix = "", initial = prefix, depth = 1)

## S3 method for class 'bru_mapper_pipe'
summary(object, ..., prefix = "", initial = prefix, depth = 1)

## S3 method for class 'bru_mapper_collect'
summary(object, ..., prefix = "", initial = prefix, depth = 1)

## S3 method for class 'summary_bru_mapper'
print(x, ...)
**Examples**

```r
mapper <-
  bru_mapper_pipe(
    list(
      bru_mapper_multi(list(
        A = bru_mapper_index(2),
        B = bru_mapper_index(3)
      )),
      bru_mapper_index(2)
    )
  )
summary(mapper, depth = 2)
```

**Description**

Print inlabru options

**Usage**

```r
## S3 method for class 'bru_options'
summary(
  object,
  legend = TRUE,
  include_global = TRUE,
  include_default = TRUE,
  ...)
```

```r
## 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
- `x` A summary_bru_options object to be printed
- `...` Further parameters, currently ignored
Examples

```r
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)
}
```

---

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**

```r
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
```
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.inla.mesh(mrsea$mesh)
  ggplot() +
  gg(mrsea$mesh) +
  gg(vrt, color = "red")
}
vertices.inla.mesh
Index

* component constructors
  component, 39
  component_list, 44
* geomes for Raster data
  gg, 74
  gg.RasterLayer, 83
  gm, 99
* geomes for inla and inlabru predictions
  gg, 74
  gg.data.frame, 75
  gg.matrix, 80
  gg.prediction, 81
  gm, 99
* geomes for meshes
  gg, 74
  gg.inla.mesh, 77
  gg.inla.mesh.1d, 79
  gm, 99
* geomes for spatial data
  gg, 74
  gg.SpatialGridDataFrame, 84
  gg.SpatialLines, 86
  gg.SpatialPixels, 88
  gg.SpatialPixelsDataFrame, 89
  gg.SpatialPoints, 90
  gg.SpatialPolygons, 92
  gm, 99
* inlabru RGL tools
  globe, 94
  glplot, 95
  glplot.inla.mesh, 96
  glplot.SpatialLines, 97
  glplot.SpatialPoints, 98
* sample generators
  generate, 71
  [.bm_list (bm_list), 6
  [.bru_like_list (like), 110
  [.bru.mapper.collect
    (bru_mapper_methods), 26
  [.bru.mapper_multi
    (bru_mapper_methods), 26
  [.component_list (component_list), 44
  $.fm_crs (fm_crs), 58
  as.bru_options (bru_options), 33
  as.bru_options(), 9
  base::.makeMessage(), 16
  bincount, 5
  bm_list, 6
  bru, 7, 12
  bru(), 5, 12, 33, 35, 39, 41, 42, 47, 71, 104,
    109–112, 118, 127, 128, 146
  bru.call_options, 9
  bru.component, 11
  bru.component (component), 39
  bru.component_list (component_list), 44
  bru.compute_linearisation, 10
  bru.convergence_plot, 12
  bru.fill_missing, 12
  bru.forward_transformation, 13
  bru.get_mapper, 14, 22, 26, 33
  bru.get_mapper_safely (bru.get_mapper), 14
  bru.info (summary.bru_info), 147
  bru.inverse_transformation
    (bru.forward_transformation), 13
  bru.log (bru.log_reset), 16
  bru.log_active (bru.log_reset), 16
  bru.log_active(), 17
  bru.log.get (bru.log_reset), 16
  bru.log.message (bru.log_reset), 16
  bru.log.message(), 35
  bru.log.reset, 16
  bru.make_stack, 18
  bru.mapper, 14, 15, 18, 26, 33
  bru.mapper.aggregate (bru.mapper), 18
  bru.mapper_collect (bru.mapper), 18
bru_mapper_index, 143
bru_mapper_const (bru_mapper), 18
bru_mapper_const (), 143
bru_mapper_define (bru_mapper), 18
bru_mapper_define (), 21
bru_mapper_factor (bru_mapper), 18
bru_mapper_generics, 22, 23, 33
bru_mapper_harmonics (bru_mapper), 18
bru_mapper_index (bru_mapper), 18
bru_mapper_linear (bru_mapper), 18
bru_mapper_matrix (bru_mapper), 18
bru_mapper_methods, 15, 22, 26, 26
bru_mapper_multi (bru_mapper), 18
bru_mapper_offset (transform), 141
bru_mapper_pipe (bru_mapper), 18
bru_mapper_scale (transform), 18
bru_mapper_taylor, 25
bru_mapper_taylor (bru_mapper), 18
bru_options_get (bru_options), 8, 33, 34, 112, 149
bru_options (), 8, 36, 112
bru_options_check (bru_options), 33
bru_options_default (bru_options), 33
bru_options_default (), 36
bru_options_get (bru_options), 33
bru_options_get (), 36
bru_options_reset (bru_options), 33
bru_options_set (bru_options), 33
bru_options_set (), 17, 109, 143
bru_rerun (bru), 7
bru_safe_inla, 37
bru_standardise_names, 37
bru_summarise, 38
bru_transformation (bru_forward_transformation), 13

c.bm_list (bm_list), 6
c.bru_mapper (bm_list), 6
c.component (component_list), 44
c.component_list (component_list), 44
class, 74
component, 39, 45
component (), 8, 13, 44, 104
component.character(), 39
component_eval, 43
component_eval (), 72, 112, 128
component_list, 42, 44
component_list.formula(), 39
countdata1 (Poisson1_1D), 123
countdata2 (Poisson2_1D), 124
countdata3a (Poisson3_1D), 125
countdata3b (Poisson3_1D), 125
cov2_1D (Poisson2_1D), 124
cprod, 46
CRS, 78, 89, 91
deltaIC, 47
devel.cvmeasure, 48
E_nc1 (Poisson1_1D), 123
E_nc2 (Poisson2_1D), 124
E_nc3a (Poisson3_1D), 125
E_nc3b (Poisson3_1D), 125
eval.spatial, 52
eval.spatial(), 143
eval.SpatialDF (transform), 141
evaluate_comp_lin, 51
evaluate_index, 51
evaluate_inputs, 52
expand_labels, 53
fm_as_inla_mesh (fm_as_spcrs), 53
fm_as_inla_mesh_segment (fm_as_spcrs),
   53
fm_as_sfc (fm_as_spcrs), 53
fm_as_spcrs, 53
fm_as_spcrs(), 65
fm_CRS, 56
fm_sparse, 58
fm_CRS(), 65, 67
fm_sparse(), 65
fm_crs.get_ellipsoid_radius
   (fm_wkt_is_geocent), 68
fm_crs.get_lengthunit
   (fm_wkt_is_geocent), 68
fm_crs.get_wkt
   (fm_wkt_is_geocent), 68
fm_crs_is_geocent (fm_wkt_is_geocent), 68
fm_crs_is_geocent (fm_wkt_is_geocent), 68
fm_crs_is_geocent (fm_wkt_is_geocent), 68
fm_crs_oblique (fm_crs), 58
fm_crs_oblique<-(fm_crs), 58
fm_crs_set_ellipsoid_radius
   (fm_wkt_is_geocent), 68
fm_crs_set_lengthunit
   (fm_wkt_is_geocent), 68
fm_crs_wkt (fm_wkt_is_geocent), 68
fm_crs_wkt (fm_wkt_is_geocent), 68
ibm_eval (bru_mapper_generics), 23
ibm_eval.bru_mapper_aggregate
 (bru_mapper_methods), 26
ibm_eval.bru_mapper_collect
 (bru_mapper_methods), 26
ibm_eval.bru_mapper_const
 (bru_mapper_methods), 26
ibm_eval.bru_mapper_logsumexp
 (bru_mapper_methods), 26
ibm_eval.bru_mapper_multi
 (bru_mapper_methods), 26
ibm_eval.bru_mapper_pipe
 (bru_mapper_methods), 26
ibm_eval.bru_mapper_scale
 (bru_mapper_methods), 26
ibm_eval.bru_mapper_taylor
 (bru_mapper_methods), 26
ibm_inla_subset (bru_mapper_generics), 23
ibm_invalid_output
 (bru_mapper_generics), 23
ibm_invalid_output(). 143
ibm_invalid_output.bru_mapper_collect
 (bru_mapper_methods), 26
ibm_invalid_output.bru_mapper_index
 (bru_mapper_methods), 26
ibm_invalid_output.bru_mapper_multi
 (bru_mapper_methods), 26
ibm_is_linear (bru Mapper Generics), 23
ibm_is_linear.bru_mapper_collect
 (bru_mapper_methods), 26
ibm_is_linear.bru_mapper_multi
 (bru_mapper_methods), 26
ibm_jacobian (bru_mapper_generics), 23
ibm_jacobian(). 143
ibm_jacobian.bru_mapper_aggregate
 (bru_mapper_methods), 26
ibm_jacobian.bru_mapper_collect
 (bru_mapper_methods), 26
ibm_jacobian.bru_mapper_const
 (bru_mapper_methods), 26
ibm_jacobian.bru_mapper_factor
 (bru_mapper_methods), 26
ibm_jacobian.bru_mapper_factor
 (bru_mapper_methods), 26
ibm_jacobian.bru_mapper_harmonics
 (bru_mapper_methods), 26
ibm_jacobian.bru_mapper_inla_mesh_1d
 (bru Mapper Methods), 26
ibm_jacobian.bru_mapper_inla_mesh_2d
 (bru Mapper Methods), 26
ibm_jacobian.bru_mapper_linear
 (bru Mapper Methods), 26
ibm_jacobian.bru_mapper_logsumexp
 (bru Mapper Methods), 26
ibm_jacobian.bru_mapper_matrix
 (bru Mapper Methods), 26
ibm_jacobian.bru_mapper_multi
 (bru Mapper Methods), 26
ibm_jacobian.bru_mapper_pipe
 (bru Mapper Methods), 26
ibm_jacobian.bru_mapper_scale
 (bru Mapper Methods), 26
ibm_jacobian.bru_mapper_taylor
 (bru Mapper Methods), 26
ibm_linear (bru Mapper Generics), 23
ibm_linear.bru_mapper_aggregate
 (bru Mapper Methods), 26
ibm_linear.bru_mapper_collect
 (bru Mapper Methods), 26
ibm_linear.bru_mapper_logsumexp
 (bru Mapper Methods), 26
ibm_linear.bru_mapper_multi
 (bru Mapper Methods), 26
ibm_linear.bru_mapper_pipe
 (bru Mapper Methods), 26
ibm_linear.bru_mapper_scale
 (bru Mapper Methods), 26
ibm_linear.bru_mapper_taylor
 (bru Mapper Methods), 26
ibm_n (bru Mapper Generics), 23
ibm_n.bru_mapper_aggregate
 (bru Mapper Methods), 26
ibm_n.bru_mapper_collect
 (bru Mapper Methods), 26
ibm_n.bru_mapper_const
 (bru Mapper Methods), 26
ibm_n.bru_mapper_factor
 (bru Mapper Methods), 26
ibm_n.bru_mapper_factor
 (bru Mapper Methods), 26
ibm_n.bru_mapper_harmonics
 (bru Mapper Methods), 26
ibm_n.bru_mapper_inla_mesh_1d
 (bru Mapper Methods), 26
ibm_n.bru_mapper_inla_mesh_2d
 (bru Mapper Methods), 26
ibm_n.bru_mapper_linear
 (bru Mapper Methods), 26
ibm_n.bru_mapper_matrix
print.summary_bru(summary.bru), 146
print.summary_bru_info
  (summary.bru_info), 147
print.summary_bru_mapper
  (summary.bru_mapper), 148
print.summary_bru_options
  (summary.bru_options), 149
pts1(Poisson1_1D), 123
pts2(Poisson2_1D), 124
pts3(Poisson3_1D), 125

robins_subset, 131
row_kron, 132

sample.lgcp, 133
seals, 104, 135
sf::st_crs(), 60, 61
shrimp, 136
sline, 137
sp::CRS(), 58
spatial.to.ppp, 138
SpatialPixels, 88
spde.posterior, 139
spoly, 140
spTransform, 143
st_check_dim, 144
st_check_polygon, 145
st_crs.fm_crs(fm_crs), 58
st_signed_area, 145
stransform, 141
summary.bru, 146
summary.bru_info, 147
summary.bru_mapper, 148
summary.bru_mapper_collect
  (summary.bru_mapper), 148
summary.bru_mapper_multi
  (summary.bru_mapper), 148
summary.bru_mapper_pipe
  (summary.bru_mapper), 148
summary.bru_options, 149
summary.component(), 146

toygroups, 104, 150

vertices.inla.mesh, 151