Package: inlabru (via r-universe)

June 4, 2026

Type Package

Title Bayesian Latent Gaussian Modelling using INLA and Extensions

Version 2.14.1.9006

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

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

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

License GPL (>= 2)

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

Encoding UTF-8

Depends methods, R (>= 4.1.0), stats

Imports generics, dplyr, fmesher (>= 0.7.0), glue, lifecycle, MatrixModels, Matrix, rlang, sf, tibble, utils, withr, Rcpp

Suggests covr, ggplot2, graphics, INLA (>= 23.01.31), knitr, maps, mgcv, patchwork, raster, RColorBrewer, rgl, rmarkdown, scales, scoringRules, shiny, sn, sp (>= 2.1), spatstat.geom, spatstat.data, sphereplot, splancs, terra (>= 1.7-66), tidyterra, testthat (>= 3.2.0), tidyr, DiagrammeR, doclisting

Enhances stars

Roxygen list(markdown = TRUE)

Config/testthat/parallel true

Config/testthat/edition 3

Collate '0_inlabru_envir.R' 'RcppExports.R' 'access_trace.R' 'access_trace_methods.R' 'bru.gof.R' 'bru.inference.R' 'bru_conversion.R' 'bru_index.R' 'deprecated.R' 'bru_input.R' 'bru_is.R' 'bru_sp.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.shrimp.R' 'data.toygroups.R' 'data.toypoints.R' 'deltaIC.R' 'effect.R' 'environment.R' 'fmesher.R' 'gcpo.R' 'ggplot.R' 'hexagon_tiling.R' 'mappers.R' 'hierarchical_basis.R' 'inla.R' 'inlabru-package.R' 'local_testthat.R' 'mapper_collect.R' 'mapper_repeat.R' 'mapper_sum.R' 'model.R' 'nlinla.R' 'object_upgrade.R' 'plotsample.R' 'pred_expr.R' 'rgl.R' 'sampling.R' 'spatstat.R' 'spde.R' 'stack.R' 'tidiers.R' 'track_plotting.R' 'transformation.R' 'used.R' 'utils.R'

VignetteBuilder knitr

BuildVignettes true

LazyData true

LazyDataCompression xz

LinkingTo Rcpp

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository https://eliaskrainski.r-universe.dev

Date/Publication 2026-06-04 07:36:27 UTC

RemoteUrl https://github.com/inlabru-org/inlabru

RemoteRef HEAD

RemoteSha 98b997b45e38bd2afd086f4c57250395f8f0ccde

Contents

`inlabru-package` *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()`. The general model specification details is documented in `bru_comp()` and `bru_obs()`. Posterior quantities beyond the basic summaries can be calculated with a `predict()` method, documented in `predict.bru()`. For point process inference `lgcp()` can be used as a shortcut to `bru(..., bru_obs(model="cp", ...))`.

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

Author(s)

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

See Also

Useful links:

- `http://www.inlabru.org`
- `https://inlabru-org.github.io/inlabru/`
- `https://github.com/inlabru-org/inlabru`
- Report bugs at `https://github.com/inlabru-org/inlabru/issues`

`as_bru_comp` *Conversion methods for* `bru_comp` *and* `bru_comp_list` *objects*

Description

Methods for converting to `bru_comp` and `bru_comp_list` objects.

Usage

```
as_bru_comp(x, ...)

as_bru_comp_list(x, ...)

## S3 method for class 'bru_comp'
as_bru_comp(x, ...)

## S3 method for class 'bru_comp_list'
as_bru_comp_list(x, ...)

## S3 method for class 'bru_comp'
as_bru_comp_list(x, ...)

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

## S3 method for class 'bru_info'
as_bru_comp_list(x, ...)

## S3 method for class 'bru_model'
as_bru_comp_list(x, ...)

## S3 method for class 'list'
as_bru_comp_list(x, ...)

## S3 method for class 'formula'
as_bru_comp_list(x, ...)
```

Arguments

`x`	An object to convert to bru_comp or bru_comp_list
`...`	Additional arguments passed on to `bru_comp_list()`.

Value

An object of class bru_comp_list.

Functions

- `as_bru_comp_list(bru)`: Extract the component list from a `bru()` object.
- `as_bru_comp_list(bru_info)`: Extract the component list from a `bru_info()` object.
- `as_bru_comp_list(bru_model)`: Extract the component list from a `bru_model()` object.

See Also

`as_bru_obs()`, `as_bru_obs_list()`

---

as_bru_mapper *Methods for mapper extraction*

Description

Extract a mapper from another object

Usage

```
as_bru_mapper(x)

## S3 method for class 'bru_mapper'
as_bru_mapper(x)

## S3 method for class 'bru_comp'
as_bru_mapper(x)

## S3 method for class 'bru_subcomp'
as_bru_mapper(x)
```

Arguments

x	Object to convert/extract

Value

A `bru_mapper` object

Examples

```
# Extract a mapper from a `bru_subcomp` object
as_bru_mapper(bru_comp("x", x, mapper = bm_index(4))$main)
```

as_bru_obs *Conversion methods for* bru_obs *and* bru_obs_list *objects*

Description

Methods for converting to bru_obs and bru_obs_list objects.

Usage

```
as_bru_obs(x, ...)

as_bru_obs_list(x, .tag = NULL)

## S3 method for class 'bru_obs'
as_bru_obs(x, ...)

## S3 method for class 'bru_obs'
as_bru_obs_list(x, .tag = NULL)

## S3 method for class 'list'
as_bru_obs_list(x, .tag = NULL)

## S3 method for class 'bru_obs_list'
as_bru_obs_list(x, .tag = NULL)

## S3 method for class 'bru'
as_bru_obs_list(x, .tag = NULL)

## S3 method for class 'bru_info'
as_bru_obs_list(x, .tag = NULL)

## S3 method for class 'bru_model'
as_bru_obs_list(x, .tag = NULL)
```

Arguments

x	An object to convert to bru_obs or bru_obs_list
...	Additional arguments passed to sub-methods.
.tag	character; optional name for the single observation model in the returned bru_obs_list object. Default is NULL, which results in automatic naming based on the tag attribute of x, if present.

Value

An object of class bru_obs or bru_obs_list.

See Also

`as_bru_comp_list()`

`augment.bru` *Augment a bru model fit with fitted values*

Description

Adds posterior mean and credible interval columns to `data`. The user must supply `pred_formula` because inlabru prediction expressions are arbitrary R and cannot be recovered from the fit object alone.

Usage

```
## S3 method for class 'bru'
augment(x, data, pred_formula, n_samples = 500L, seed = NULL, ...)
```

Arguments

`x`	A fitted `bru` object.
`data`	A data frame of covariate values.
`pred_formula`	A formula passed to `inlabru::predict()`.
`n_samples`	Number of posterior samples (default 500).
`seed`	Optional integer. If non-`NULL`, the session RNG is reset to this value before `inlabru::predict()` is called so that two identical calls return identical posterior summaries. `NULL` leaves the session RNG state unchanged (predictions then vary run-to-run because `inlabru::predict()` samples from the posterior).
`...`	Passed to `inlabru::predict()`.

Value

`data` with additional columns `.fitted`, `.fitted_low`, `.fitted_high`, `.fitted_sd`.

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 intervals per bin. Increase this if the bins are wide and the LGCP is not smooth.
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) && require(fmesher) && bru_safe_inla()) {
  # 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.out = 50)
  mesh1D <- fmesher::fm_mesh_1d(x, boundary = "free")
  matern <- INLA::inla.spde2.pcmatern(mesh1D,
    prior.range = c(1, 0.01),
    prior.sigma = c(1, 0.01),
    constr = TRUE
  )
  mdl <- x ~ spde1D(x, model = matern) + Intercept(1)
  fit.spde <- lgcp(mdl, pts2, domain = list(x = mesh1D))

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

  # Plot them!
  attributes(bc)$ggp
}

## End(Not run)
```

---

bm_aggregate *Mapper for aggregation*

Description

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.

Usage

```
bm_aggregate(rescale = FALSE, n_block = NULL, type = NULL)

bru_mapper_aggregate(...)
```

Arguments

rescale	logical; For bm_aggregate and bm_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. The priority order is input$n_block, the mapper definition n_block, then max(input$block).
type	character; if non-NULL, overrides the rescale argument, and constructs an aggregation mapper of the given type instead. Supported values are "sum", "average" (for regular bm_aggregate()), "logsumexp", "logaverageexp" (for bm_logsumexp()), and "logitaverage" (for bm_logitaverage()).
...	Arguments passed on to bm_aggregate()

Value

A bm_aggregate mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: bm_collect(), bm_const(), bm_factor(), bm_fm_mesh_1d, bm_fmesher(), bm_harmonics(), bm_index(), bm_linear(), bm_logitaverage(), bm_logsumexp(), bm_marginal(), bm_matrix(), bm_multi(), bm_pipe(), bm_reparam(), bm_repeat(), bm_scale(), bm_shift(), bm_sum(), bm_taylor(), bru_get_mapper(), bru_mapper()

Examples

```
m <- bm_aggregate()
ibm_eval2(m, list(block = c(1, 2, 1, 2), weights = 1:4), 11:14)
ibm_eval2(m, list(block = c(1, 2, 1, 2), weights = 1:4, n_block = 3), 11:14)
```

bm_collect *Mapper for concatenated variables*

Description

Constructs a concatenated collection mapping

Usage

```
bm_collect(mappers, hidden = FALSE)

bru_mapper_collect(...)

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

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

Arguments

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
...	Arguments passed on to bm_scale()
x	object from which to extract element(s)
i	indices specifying element(s) to extract
drop	logical; For [.bm_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 bm_collect object). Default: TRUE

Value

- [-indexing a bm_collect extracts a subset bm_collect object (for drop FALSE) or an individual sub-mapper (for drop TRUE, and i identifies a single element)

See Also

bru_mapper, bru_mapper_generics

Other mappers: bm_aggregate(), bm_const(), bm_factor(), bm_fm_mesh_1d, bm_fmesher(), bm_harmonics(), bm_index(), bm_linear(), bm_logitaverage(), bm_logsumexp(), bm_marginal(), bm_matrix(), bm_multi(), bm_pipe(), bm_reparam(), bm_repeat(), bm_scale(), bm_shift(), bm_sum(), bm_taylor(), bru_get_mapper(), bru_mapper()

Examples

```
(m <- bm_collect(list(
  a = bm_index(2),
  b = bm_index(3)
), hidden = FALSE))
ibm_eval2(m, list(a = c(1, 2), b = c(1, 3, 2)), 1:5)
```

`bm_const` *Constant mapper*

Description

Create a constant mapper

Usage

```
bm_const()

bru_mapper_const()
```

Value

A `bm_const` mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: `bm_aggregate()`, `bm_collect()`, `bm_factor()`, `bm_fm_mesh_1d`, `bm_fmesher()`, `bm_harmonics()`, `bm_index()`, `bm_linear()`, `bm_logitaverage()`, `bm_logsumexp()`, `bm_marginal()`, `bm_matrix()`, `bm_multi()`, `bm_pipe()`, `bm_reparam()`, `bm_repeat()`, `bm_scale()`, `bm_shift()`, `bm_sum()`, `bm_taylor()`, `bru_get_mapper()`, `bru_mapper()`

Examples

```
m <- bm_const()
ibm_eval2(m, input = 1:4)
```

bm_factor *Mapper for factor variables*

Description

Create a factor mapper

Usage

```
bm_factor(values, factor_mapping, indexed = FALSE)

bru_mapper_factor(...)
```

Arguments

values	Input values calculated by bru_input.bru_input()
factor_mapping	character; selects the type of factor mapping. • 'contrast' for leaving out the first factor level. • 'full' for keeping all levels.
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.
...	Arguments passed on to bm_factor()

Value

A bm_factor mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: bm_aggregate(), bm_collect(), bm_const(), bm_fm_mesh_1d, bm_fmesher(), bm_harmonics(), bm_index(), bm_linear(), bm_logitaverage(), bm_logsumexp(), bm_marginal(), bm_matrix(), bm_multi(), bm_pipe(), bm_reparam(), bm_repeat(), bm_scale(), bm_shift(), bm_sum(), bm_taylor(), bru_get_mapper(), bru_mapper()

Examples

```
m <- bm_factor(factor(c("a", "b")), "full")
ibm_eval2(m, input = c("b", "a", "a", "b"), state = c(1, 3))

m <- bm_factor(factor(c("a", "b")), "contrast")
ibm_eval2(m, input = factor(c("b", "a", "a", "b")), state = 2)
```

bm_fm_mesh_1d *Mapper for* fm_mesh_1d

Description

Create mapper for an fm_mesh_1d object

Usage

```
## S3 method for class 'fm_mesh_1d'
bru_mapper(mesh, indexed = TRUE, ...)
```

Arguments

mesh	An fm_mesh_1d object to use as a mapper
indexed	logical; If TRUE (default), the ibm_values() output will be the integer indexing sequence for the latent variables (needed for spde models). If FALSE, points representative of the basis centres are returned (useful for an interpolator for rw2 models and similar, for fmesher versions >= 0.3.0.9002).
...	Arguments passed on to bm_fmesher()

Value

A bm_fm_mesh_1d or bm_fmesher object. The the general bm_fmesher() mapper handles all indexed fmesher objects.

See Also

bru_mapper, bru_mapper_generics

Other mappers: bm_aggregate(), bm_collect(), bm_const(), bm_factor(), bm_fmesher(), bm_harmonics(), bm_index(), bm_linear(), bm_logitaverage(), bm_logsumexp(), bm_marginal(), bm_matrix(), bm_multi(), bm_pipe(), bm_reparam(), bm_repeat(), bm_scale(), bm_shift(), bm_sum(), bm_taylor(), bru_get_mapper(), bru_mapper()

Examples

```
m <- bru_mapper(fmesher::fm_mesh_1d(c(1:3, 5, 7)))
ibm_values(m)
ibm_eval(m, 1:7, 1:5)

m <- bru_mapper(fmesher::fm_mesh_1d(c(1:3, 5, 7)), indexed = FALSE)
ibm_values(m)
ibm_eval(m, 1:7, 1:5)

m <- bru_mapper(
  fmesher::fm_mesh_1d(c(1:3, 5, 7), degree = 2, boundary = "free"),
  indexed = FALSE
)
```

```
ibm_values(m)
ibm_eval(m, 1:7, 1:6)
```

---

bm_fmesher *Mapper for general* fmesher *function space objects*

Description

Creates a mapper for general fmesher function space objects.

Usage

```
bm_fmesher(mesh)

bru_mapper_fmesher(...)

## S3 method for class 'fm_mesh_2d'
bru_mapper(mesh, ...)
```

Arguments

mesh	An fmesher object to map, supported by fmesher::fm_basis(mesh, input) and fmesher::fm_dof(mesh).
...	Arguments passed on to bm_fmesher()

Details

Handles indexed mapping for all fmesher classes that support fm_dof() and fm_basis() methods. For non-indexed mapping of fm_mesh_1d objects, use bru_mapper(mesh, indexed = FALSE) which invokes the bru_mapper.fm_mesh_1d() method.

Value

A bm_fmesher object.

Functions

- bru_mapper(fm_mesh_2d): Equivalent to calling bm_fmesher(). Note: Prior to version 2.12.0.9021, this returned a bru_mapper_fm_mesh_2d object. Also see the note for bru_mapper.fm_mesh_1d().

See Also

bru_mapper, bru_mapper_generics

Other mappers: bm_aggregate(), bm_collect(), bm_const(), bm_factor(), bm_fm_mesh_1d, bm_harmonics(), bm_index(), bm_linear(), bm_logitaverage(), bm_logsumexp(), bm_marginal(), bm_matrix(), bm_multi(), bm_pipe(), bm_reparam(), bm_repeat(), bm_scale(), bm_shift(), bm_sum(), bm_taylor(), bru_get_mapper(), bru_mapper()

Examples

```
m <- bm_fmesher(fmesher::fmexample$mesh)
ibm_n(m)
ibm_eval(m, as.matrix(expand.grid(-2:2, -2:2)), seq_len(ibm_n(m)))
```

---

bm_harmonics *Mapper for cos/sin functions*

Description

Constructs a mapper for cos/sin functions of orders 1 (if `intercept` is `TRUE`, otherwise 0) through `order`. The total number of basis functions is `intercept + 2 * order`.

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

```
scaling = 1 / (1 + (0:4)^2)
x <- seq(0, 1, length.out = 11)
bmh1 = bm_harmonics(order = 4, interval = c(0, 1))
u1 <- ibm_eval(
  bmh1,
  input = x,
  state = rnorm(9, sd = rep(scaling, c(1, 2, 2, 2, 2)))
)
```

Then, with

```
bmh2 = bm_harmonics(order = 4, scaling = scaling)
u2 = ibm_eval(bmh2, input = x, state = rnorm(9))
```

the stochastic properties of `u1` and `u2` will be the same, with `scaling^2` determining the variance for each frequency contribution.

The period for the first order harmonics is shifted and scaled to match `interval`.

Usage

```
bm_harmonics(order = 1, scaling = 1, intercept = TRUE, interval = c(0, 1))

bru_mapper_harmonics(...)
```

Arguments

order	For bm_harmonics, specifies the maximum cos/sin order. (Default 1)
scaling	For bm_harmonics, specifies an optional vector of scaling factors of length intercept + order, or a common single scalar.
intercept	logical; For bm_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).
...	Arguments passed on to bm_harmonics()

Value

A bm_harmonics mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: bm_aggregate(), bm_collect(), bm_const(), bm_factor(), bm_fm_mesh_1d, bm_fmesher(), bm_index(), bm_linear(), bm_logitaverage(), bm_logsumexp(), bm_marginal(), bm_matrix(), bm_multi(), bm_pipe(), bm_reparam(), bm_repeat(), bm_scale(), bm_shift(), bm_sum(), bm_taylor(), bru_get_mapper(), bru_mapper()

Examples

```
m <- bm_harmonics(2)
ibm_eval2(m, input = c(0, pi / 4, pi / 2, 3 * pi / 4), 1:5)
```

---

bm_index *Mapper for indexed variables*

Description

Create an indexing mapper

Usage

```
bm_index(n = 1L)

bru_mapper_index(...)
```

Arguments

n	Size of a model for bm_index
...	Arguments passed on to bm_index()

Value

A bm_index mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: bm_aggregate(), bm_collect(), bm_const(), bm_factor(), bm_fm_mesh_1d, bm_fmesher(), bm_harmonics(), bm_linear(), bm_logitaverage(), bm_logsumexp(), bm_marginal(), bm_matrix(), bm_multi(), bm_pipe(), bm_reparam(), bm_repeat(), bm_scale(), bm_shift(), bm_sum(), bm_taylor(), bru_get_mapper(), bru_mapper()

Examples

```
m <- bm_index(4)
ibm_eval(m, -2:6, 1:4)
```

---

bm_linear *Mapper for a linear effect*

Description

Create a mapper for linear effects

Usage

```
bm_linear()

bru_mapper_linear()
```

Value

A bm_linear mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: bm_aggregate(), bm_collect(), bm_const(), bm_factor(), bm_fm_mesh_1d, bm_fmesher(), bm_harmonics(), bm_index(), bm_logitaverage(), bm_logsumexp(), bm_marginal(), bm_matrix(), bm_multi(), bm_pipe(), bm_reparam(), bm_repeat(), bm_scale(), bm_shift(), bm_sum(), bm_taylor(), bru_get_mapper(), bru_mapper()

Examples

```
m <- bm_linear()
ibm_eval(m, input = 1:4, state = 2)
```

bm_list *Methods for mapper lists*

Description

bru_mapper lists can be combined into bm_list lists.

Usage

```
as_bm_list(x)

## S3 method for class 'list'
as_bm_list(x)

## S3 method for class 'bm_list'
as_bm_list(x)

## S3 method for class 'bru_comp_list'
as_bm_list(x)

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

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

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

Arguments

x	bm_list object from which to extract element(s)
...	Objects to be combined.
i	indices specifying elements to extract

Value

A bm_list object

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 = bm_const(), B = bm_scale())
str(m)
str(m[2])
```

`bm_logitaverage` *Mapper for logit-sum-inverse-logit aggregation*

Description

[Experimental] Constructs a mapper that averages elements of `plogis(state)`, with optional non-negative weighting, and then takes the `qlogis()`. Relies on the input handling methods for `bm_aggregate`. To avoid numerical issues, it uses `plogis(x, log.p = TRUE)`, `plogis(-x, log.p = TRUE)`, and the equivalent of two applications of `bm_logsumexp()` to evaluate $\log(p_k) - \log(1 - p_k)$, where

$p_k = \sum_{i \in I_k} w_i / (1 + e^{-\eta_i}) / \sum_{i \in I_k} w_i$

Usage

```
bm_logitaverage(n_block = NULL)
```

Arguments

`n_block`	Predetermined number of output blocks. If `NULL`, overrides the maximum block index in the inputs. The priority order is `input$n_block`, the mapper definition `n_block`, then `max(input$block)`.

Value

A `bm_logitaverage/bm_aggregate` mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: `bm_aggregate()`, `bm_collect()`, `bm_const()`, `bm_factor()`, `bm_fm_mesh_1d`, `bm_fmesher()`, `bm_harmonics()`, `bm_index()`, `bm_linear()`, `bm_logsumexp()`, `bm_marginal()`, `bm_matrix()`, `bm_multi()`, `bm_pipe()`, `bm_reparam()`, `bm_repeat()`, `bm_scale()`, `bm_shift()`, `bm_sum()`, `bm_taylor()`, `bru_get_mapper()`, `bru_mapper()`

Examples

```
m <- bm_logitaverage()
ibm_eval2(m, list(block = c(1, 2, 1, 2), weights = 1:4), 11:14)
ibm_eval2(m, list(block = c(1, 2, 1, 2), weights = 1:4, n_block = 3), 11:14)
```

bm_logsumexp *Mapper for log-sum-exp aggregation*

Description

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 $v_k = \log[\sum_{i \in I_k} w_i \exp(u_i) / \sum_{i \in I_k} w_i]$ calculations. Relies on the input handling methods for `bm_aggregate`, but also allows the weights to be supplied on a logarithmic scale as `log_weights`. To avoid numerical overflow, it uses the common method of internally shifting the state blockwise; $v_k = s_k + \log[\sum_{i \in I_k} \exp(u_i + \log(w_i) - s_k)]$, where $s_k = \max_{i \in I_k} u_i + \log(w_i)$ is the shift for block k.

Usage

```
bm_logsumexp(rescale = FALSE, n_block = NULL)

bru_mapper_logsumexp(...)
```

Arguments

`rescale`	logical; For `bm_aggregate` and `bm_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. The priority order is `input$n_block`, the mapper definition `n_block`, then `max(input$block)`.
`...`	Arguments passed on to `bm_logsumexp()`

Value

A `bm_logsumexp/bm_aggregate` mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: `bm_aggregate()`, `bm_collect()`, `bm_const()`, `bm_factor()`, `bm_fm_mesh_1d`, `bm_fmesher()`, `bm_harmonics()`, `bm_index()`, `bm_linear()`, `bm_logitaverage()`, `bm_marginal()`, `bm_matrix()`, `bm_multi()`, `bm_pipe()`, `bm_reparam()`, `bm_repeat()`, `bm_scale()`, `bm_shift()`, `bm_sum()`, `bm_taylor()`, `bru_get_mapper()`, `bru_mapper()`

Examples

```
m <- bm_logsumexp()
ibm_eval2(m, list(block = c(1, 2, 1, 2), weights = 1:4), 11:14)
ibm_eval2(m, list(block = c(1, 2, 1, 2), weights = 1:4, n_block = 3), 11:14)
```

`bm_marginal` *Mapper for marginal distribution transformation*

Description

Constructs a mapper that transforms the marginal distribution `state` from $\mathrm{N}(0, 1)$ to the distribution of a given (continuous) quantile function. The `...` arguments are used as parameter arguments to `qfun`, `pfun`, `dfun`, and `dqfun`.

Usage

```
bm_marginal(qfun, pfun = NULL, dfun = NULL, dqfun = NULL, ..., inverse = FALSE)

bru_mapper_marginal(...)
```

Arguments

`qfun`	A quantile function, supporting arguments `p`, `lower.tail`, and `log.p`, like `stats::qnorm()`.
`pfun`	A CDF, supporting arguments `q`, `lower.tail`, and `log.p`, like `stats::pnorm()`. Only needed and used when `xor(mapper[["inverse"]], reverse)` is `TRUE` in a method call. Default `NULL`
`dfun`	A pdf, supporting arguments `x` and `log`, like `stats::dnorm()`. If `NULL` (default), uses finite differences on `qfun` or `pfun` instead.
`dqfun`	A function evaluating the reciprocal of the derivative of `qfun`, i.e. the density at a given CDF value. If `NULL` (default), uses `dfun(qfun(...),...)` or finite differences on `qfun` or `pfun` instead. Must support the same arguments as `qfun`.
`...`	Arguments passed on to `bm_marginal()`
`inverse`	logical; If `FALSE` (default), `bm_marginal()` defines a mapping from standard Normal to a specified distribution. If `TRUE`, it defines a mapping from the specified distribution to a standard Normal.

Value

A `bm_marginal` mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: `bm_aggregate()`, `bm_collect()`, `bm_const()`, `bm_factor()`, `bm_fm_mesh_1d`, `bm_fmesher()`, `bm_harmonics()`, `bm_index()`, `bm_linear()`, `bm_logitaverage()`, `bm_logsumexp()`, `bm_matrix()`, `bm_multi()`, `bm_pipe()`, `bm_reparam()`, `bm_repeat()`, `bm_scale()`, `bm_shift()`, `bm_sum()`, `bm_taylor()`, `bru_get_mapper()`, `bru_mapper()`

Examples

```
m <- bm_marginal(qexp, pexp, rate = 1 / 8)
(val <- ibm_eval(m, state = -5:5))
ibm_eval(m, state = val, reverse = TRUE)
m <- bm_marginal(qexp, pexp, dexp, rate = 1 / 8)
ibm_eval2(m, state = -3:3)
```

`bm_matrix` *Mapper for matrix multiplication*

Description

Create a matrix mapper, for a given number of columns

Usage

```
bm_matrix(labels)

bru_mapper_matrix(...)
```

Arguments

`labels`	Column labels for matrix mappings; Can be factor, character, or a single integer specifying the number of columns for integer column indexing.
`...`	Arguments passed on to `bm_matrix()`

Value

A `bm_matrix` mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: `bm_aggregate()`, `bm_collect()`, `bm_const()`, `bm_factor()`, `bm_fm_mesh_1d`, `bm_fmesher()`, `bm_harmonics()`, `bm_index()`, `bm_linear()`, `bm_logitaverage()`, `bm_logsumexp()`, `bm_marginal()`, `bm_multi()`, `bm_pipe()`, `bm_reparam()`, `bm_repeat()`, `bm_scale()`, `bm_shift()`, `bm_sum()`, `bm_taylor()`, `bru_get_mapper()`, `bru_mapper()`

Examples

```
m <- bm_matrix(labels = c("a", "b"))
ibm_values(m)
ibm_eval2(m, input = matrix(1:6, 3, 2), state = 2:3)

m <- bm_matrix(labels = 2L)
ibm_values(m)
ibm_eval2(m, input = matrix(1:6, 3, 2), state = 2:3)
```

bm_multi *Mapper for tensor product domains*

Description

Constructs a row-wise Kronecker product mapping of linear/affine mappers. Any offset in sub-mappers is added into a combined offset. Only linear/affine sub-mappers are allowed.

Usage

```
bm_multi(mappers, simplify = FALSE)

bru_mapper_multi(...)

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

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

Arguments

mappers	A list of bru_mapper objects
simplify	logical; If TRUE, removes trivial submappers. Currently only sub-mappers of class bm_index() with ibm_n() == 1L are removed, and only if the mappers are named (to avoid ordering mismatches). Default: FALSE
...	Arguments passed on to bm_multi()
x	object from which to extract element(s)
i	indices specifying element(s) to extract
drop	logical; For [.bm_multi, 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 bm_multi object). Default: TRUE

Value

A bm_multi mapper object.

- [-indexing a bm_multi extracts a subset bm_multi object (for drop FALSE) or an individual sub-mapper (for drop TRUE, and i identifies a single element)

See Also

bru_mapper, bru_mapper_generics

Other mappers: bm_aggregate(), bm_collect(), bm_const(), bm_factor(), bm_fm_mesh_1d, bm_fmesher(), bm_harmonics(), bm_index(), bm_linear(), bm_logitaverage(), bm_logsumexp(), bm_marginal(), bm_matrix(), bm_pipe(), bm_reparam(), bm_repeat(), bm_scale(), bm_shift(), bm_sum(), bm_taylor(), bru_get_mapper(), bru_mapper()

Examples

```
(m <- bm_multi(list(a = bm_index(2), b = bm_index(3))))
ibm_eval2(m, list(a = c(1, 2, 1), b = c(1, 3, 2)), 1:6)
```

---

bm_pipe *Mapper for linking several mappers in sequence*

Description

Create a pipe mapper, where `mappers` is a list of mappers, and the evaluated output of each mapper is handed as the state to the next mapper. The `input` format for the `ibm_eval` and `ibm_jacobian` methods is a list of inputs, one for each mapper.

Usage

```
bm_pipe(mappers)

bru_mapper_pipe(...)
```

Arguments

`mappers`	A list of `bru_mapper` objects
`...`	Arguments passed on to `bm_pipe()`

Value

A `bm_pipe` mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: `bm_aggregate()`, `bm_collect()`, `bm_const()`, `bm_factor()`, `bm_fm_mesh_1d`, `bm_fmesher()`, `bm_harmonics()`, `bm_index()`, `bm_linear()`, `bm_logitaverage()`, `bm_logsumexp()`, `bm_marginal()`, `bm_matrix()`, `bm_multi()`, `bm_reparam()`, `bm_repeat()`, `bm_scale()`, `bm_shift()`, `bm_sum()`, `bm_taylor()`, `bru_get_mapper()`, `bru_mapper()`

Examples

```
m <- bm_pipe(list(
  scale = bm_scale(),
  shift = bm_shift()
))
ibm_eval2(m, input = list(scale = 2, shift = 1:4), state = 1:4)
```

bm_reparam *Mapper for reparameterising mapper states*

Description

Creates a mapper for handling basis conversions. Functionally equivalent to `bm_pipe(list(bm_matrix(ncol(B)), mapper))`, but with an internally stored matrix input `B` for efficiency, and allowing the mapper `input` format to be identical to that of the original `mapper`.

Usage

```
bm_reparam(mapper, B)
```

Arguments

`mapper`	A `bru_mapper` object
`B`	a square or rectangular basis conversion matrix

Value

A `bm_reparam` mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: `bm_aggregate()`, `bm_collect()`, `bm_const()`, `bm_factor()`, `bm_fm_mesh_1d`, `bm_fmesher()`, `bm_harmonics()`, `bm_index()`, `bm_linear()`, `bm_logitaverage()`, `bm_logsumexp()`, `bm_marginal()`, `bm_matrix()`, `bm_multi()`, `bm_pipe()`, `bm_repeat()`, `bm_scale()`, `bm_shift()`, `bm_sum()`, `bm_taylor()`, `bru_get_mapper()`, `bru_mapper()`

Examples

```
# 2->2 reparameterisation; (u1,u2) -> (u1, u1 + u2)
(m <- bm_reparam(bm_index(2), B = matrix(c(1, 1, 0, 1), 2, 2)))

# 2->3 reparameterisation; (u1,u2) -> (u1, u2, u1+u2)
# This is an example of a low-rank representation of a higher-dimensional
# state vector.
(m <- bm_reparam(bm_index(3), B = cbind(c(1, 0, 1), c(0, 1, 1))))
```

bm_repeat *Mapper for repeating a mapper*

Description

Defines a repeated-space mapper that sums the contributions for each copy. The `ibm_n()` method returns `ibm_n(mapper) * n_rep`, and `ibm_values()` returns `seq_len(ibm_n(mapper))`.

Usage

```
bm_repeat(mapper, n_rep, interleaved = FALSE)

bru_mapper_repeat(...)
```

Arguments

`mapper`	The mapper to be repeated.
`n_rep`	The number of times to repeat the mapper. If a vector, the non-interleaved repeats are combined into a single repeat mapping, and combined with interleaved repeats via a `bm_sum()` of mappers.
`interleaved`	logical; if `TRUE`, the repeated mapping columns are interleaved; `(x1[1], x2[1], ..., x1[2], x2[2], .` If `FALSE` (default), the repeated mapping columns are contiguous, `(x1[1], x1[2], ..., x2[1], x2[2],` and the Jacobian is a `cbind()` of the Jacobians of the repeated mappers. If `n_rep` is a vector, `interleaved` should either be a single logical, or a vector of the same length. Each element applies to the corresponding `n_rep` repetition specification.
`...`	Arguments passed on to `bm_scale()`

Value

A `bm_repeat` or `bm_sum` object, or the original input `mapper`.

See Also

bru_mapper, bru_mapper_generics

Other mappers: `bm_aggregate()`, `bm_collect()`, `bm_const()`, `bm_factor()`, `bm_fm_mesh_1d`, `bm_fmesher()`, `bm_harmonics()`, `bm_index()`, `bm_linear()`, `bm_logitaverage()`, `bm_logsumexp()`, `bm_marginal()`, `bm_matrix()`, `bm_multi()`, `bm_pipe()`, `bm_reparam()`, `bm_scale()`, `bm_shift()`, `bm_sum()`, `bm_taylor()`, `bru_get_mapper()`, `bru_mapper()`

Examples

```
(m0 <- bm_index(3))
(m <- bm_repeat(m0, 5))
ibm_n(m)
ibm_values(m)
ibm_jacobian(m, 1:3)
```

```
ibm_eval(m, 1:3, seq_len(ibm_n(m)))

# Interleaving and grouping
(m <- bm_repeat(m0, c(2, 1, 2), c(TRUE, FALSE, FALSE)))
ibm_n(m)
ibm_values(m)
ibm_jacobian(m, 1:3)
ibm_eval(m, 1:3, seq_len(ibm_n(m)))
```

`bm_scale` *Mapper for element-wise scaling*

Description

Create a standalone scaling mapper that can be used as part of a `bm_pipe`. If `mapper` is non-null, the `bm_scale()` constructor returns `bm_pipe(list(mapper = mapper, scale = bm_scale()))`

Usage

```
bm_scale(mapper = NULL)

bru_mapper_scale(...)
```

Arguments

`mapper`	An optional `bru_mapper` to be scaled.
`...`	Arguments passed on to `bm_scale()`

Value

A `bm_scale` mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: `bm_aggregate()`, `bm_collect()`, `bm_const()`, `bm_factor()`, `bm_fm_mesh_1d`, `bm_fmesher()`, `bm_harmonics()`, `bm_index()`, `bm_linear()`, `bm_logitaverage()`, `bm_logsumexp()`, `bm_marginal()`, `bm_matrix()`, `bm_multi()`, `bm_pipe()`, `bm_reparam()`, `bm_repeat()`, `bm_shift()`, `bm_sum()`, `bm_taylor()`, `bru_get_mapper()`, `bru_mapper()`

Examples

```
m <- bm_scale()
ibm_eval2(m, c(1, 2, 1, 2), 1:4)
```

bm_shift *Mapper for element-wise shifting*

Description

Create a standalone shift mapper that can be used as part of a bm_pipe. If mapper is non-null, the bm_shift() constructor returns bm_pipe(list(mapper = mapper, shift = bm_shift()))

Usage

```
bm_shift(mapper = NULL)

bru_mapper_shift(...)
```

Arguments

mapper	If non-NULL, a bru_mapper to be shifted.
...	Arguments passed on to bm_shift()

Value

A bm_shift mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: bm_aggregate(), bm_collect(), bm_const(), bm_factor(), bm_fm_mesh_1d, bm_fmesher(), bm_harmonics(), bm_index(), bm_linear(), bm_logitaverage(), bm_logsumexp(), bm_marginal(), bm_matrix(), bm_multi(), bm_pipe(), bm_reparam(), bm_repeat(), bm_scale(), bm_sum(), bm_taylor(), bru_get_mapper(), bru_mapper()

Examples

```
m <- bm_shift()
ibm_eval2(m, c(1, 2, 1, 2), 1:4)
```

bm_sum *Mapper for adding multiple mappers*

Description

Defines a mapper that adds the effects of each submapper. The `ibm_n()` method returns the sum of `ibm_n(mappers[[k]])`, and `ibm_values()` returns `seq_len(ibm_n(mapper))`.

Usage

```
bm_sum(mappers, single_input = FALSE)

bru_mapper_sum(...)

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

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

Arguments

`mappers`	A list of `bru_mapper` objects.
`single_input`	logical. If `TRUE`, the input is passed to all sub-mappers. Otherwise, the input should be a list, data.frame, or matrix. If the `mappers` list has named entries, the `input` can reference their corresponding sub-mapper using its name.
`...`	Arguments passed on to `bm_sum()`
`x`	object from which to extract element(s)
`i`	indices specifying element(s) to extract
`drop`	logical; For `[.bm_sum`, 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 `bm_sum` object). Default: `TRUE`

Value

A `bm_sum` object.

- `[`-indexing a `bm_sum` extracts a subset `bm_sum` object (for drop `FALSE`) or an individual sub-mapper (for drop `TRUE`, and `i` identifies a single element)

See Also

bru_mapper, bru_mapper_generics

Other mappers: `bm_aggregate()`, `bm_collect()`, `bm_const()`, `bm_factor()`, `bm_fm_mesh_1d`, `bm_fmesher()`, `bm_harmonics()`, `bm_index()`, `bm_linear()`, `bm_logitaverage()`, `bm_logsumexp()`, `bm_marginal()`, `bm_matrix()`, `bm_multi()`, `bm_pipe()`, `bm_reparam()`, `bm_repeat()`, `bm_scale()`, `bm_shift()`, `bm_taylor()`, `bru_get_mapper()`, `bru_mapper()`

Examples

```
(m <- bm_sum(list(a = bm_index(3), b = bm_index(2))))
ibm_n(m)
ibm_values(m)
ibm_jacobian(m, list(a = 1:3, b = c(1, 1, 2)))
ibm_eval(
  m,
  list(a = 1:3, b = c(1, 1, 2)),
  seq_len(ibm_n(m))
)
```

---

bm_taylor *Mapper for linear Taylor approximations*

Description

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)`

Usage

```
bm_taylor(offset = NULL, jacobian = NULL, state0 = NULL, values_mapper = NULL)

bru_mapper_taylor(...)
```

Arguments

`offset`	For `bm_taylor`, an offset vector giving the value of the linearisation at `state0`. May be `NULL`, interpreted as an all-zero vector of length determined by a non-null Jacobian.
`jacobian`	For `bm_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 `bm_taylor`, the reference `state` for the linearisation, or a list of such states 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)
`...`	Arguments passed on to `bm_taylor()`

Value

A `bm_taylor` mapper object.

See Also

bru_mapper, bru_mapper_generics

Other mappers: bm_aggregate(), bm_collect(), bm_const(), bm_factor(), bm_fm_mesh_1d, bm_fmesher(), bm_harmonics(), bm_index(), bm_linear(), bm_logitaverage(), bm_logsumexp(), bm_marginal(), bm_matrix(), bm_multi(), bm_pipe(), bm_reparam(), bm_repeat(), bm_scale(), bm_shift(), bm_sum(), bru_get_mapper(), bru_mapper()

Examples

```
m <- bm_taylor(
  offset = rep(2, 3),
  jacobian = matrix(1:6, 3, 2),
  state0 = c(1, 2)
)
ibm_eval2(m, state = 2:3)
```

---

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 observation models is straightforward. See bru_obs() for more information on how to provide additional models 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())

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

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

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

Arguments

components	Latent component definitions, either as a bru_comp_list() object, or a formula-like specification. Also used to define a default linear additive predictor. See bru_comp() for details.
...	Observation models, each constructed by a calling bru_obs(), or bru_obs_list(). Alternatively, for backwards compatibility, may be named parameters that can be passed to a single bru_obs() call. These arguments will be evaluated before calling bru_obs(), in order to detect if they already are bru_obs objects. This means that special arguments that are only available in the context of data or response_data (such as Ntrials) will only work properly in direct calls to bru_obs().
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
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
x	An object to be printed

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.

Methods (by generic)

- summary(bru): Takes a fitted bru object produced by bru() or lgcp() and creates various summaries from it, including the summary output from the corresponding INLA::sumary() method. The ... arguments are passed on to component summary functions, see summary.bru_comp().
- print(bru): Print a summary of a bru object.

Functions

- bru_rerun(): Continue the optimisation from a previously computed estimate. The estimation options list can be given new values to override the original settings.

  To rerun with a subset of the data (e.g. for cross validation or prior sampling), use bru_set_missing() to set all or part of the response data to NA before calling bru_rerun().

Author(s)

Fabian E. Bachl <bachlfab@gmail.com>

Examples

```
if (bru_safe_inla()) {
  # 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(1), family = "gaussian", data = input.df)

  # Obtain summary
  fit$summary.fixed
}


if (bru_safe_inla()) {
  # Alternatively, we can use the bru_obs() function to construct the
  # likelihood:

  lik <- bru_obs(
    family = "gaussian",
    formula = y ~ x + Intercept,
    data = input.df
  )
  fit <- bru(~ x + Intercept(1), lik)
  fit$summary.fixed
}

# An important addition to the INLA methodology is bru's ability to use
# non-linear predictors. Such a predictor can be formulated via bru_obs()'s
# \code{formula} parameter. The z(1) notation is needed to ensure that
# the z component should be interpreted as single latent variable and not
# a covariate:

if (bru_safe_inla()) {
  z <- 2
  input.df <- within(input.df, {
    y <- 5 + exp(z) * x + rnorm(10, mean = 0, sd = 0.1)
  })
  lik <- bru_obs(
    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_block_gcpo` *Extract block-averaged GCPO scores from one or more fitted bru models*

Description

After fitting a model with `control.gcpo = list(enable = TRUE)`, this function reads the raw per-observation GCPO vector from `fit$gcpo$gcpo` and returns one score per block or observation group.

Two cases are handled automatically:

Block-based (lgcp / `"cp"` family) When `BRU_block` is present in `response_data` (populated automatically by the `"cp"` family machinery from the samplers structure), INLA repeats the same GCPO value for every observation in a leave-out block. The mean within each block collapses the repeated values to one score per block.

Groups-based (all other families) When `BRU_block` is absent, INLA builds groups internally (via `num.level.sets` or `friends`) or from a user-supplied `groups` list. One score per observation is returned directly from `fit$gcpo$gcpo`.

Scores are returned on the probability scale, consistent with `fit$cpo$cpo`. For multi-likelihood models, cumulative row offsets into the stacked INLA response vector are handled automatically.

Usage

```
bru_block_gcpo(fit, ...)
```

Arguments

`fit`	A fitted object of class `bru`, or a named list of such objects, or several such objects passed via `\dots`. Each must have been fitted with `control.gcpo = list(enable = TRUE)` so that `fit$gcpo$gcpo` is non-NULL.
`...`	Additional named fitted `bru` objects.

Value

For a single fit, a list with two elements:

`blocks` Named list with one element per likelihood, each a vector of block labels (for `"cp"` models: unique `BRU_block` values; for other models: observation indices).

`gcpo` GCPO scores on the probability scale, consistent with `fit$cpo$cpo`. A numeric vector for single-likelihood models; a named list of numeric vectors for multi-likelihood models.

For multiple fits, a named list of such objects, one per fit.

See Also

`bru()`, `bru_obs()`, `bru_obs_control_gcpo()`, `bru_gcpo_table()`

Examples

```
if (bru_safe_inla()) {
  # Block-based example (lgcp)
  cvpart <- cv_hex(
    gorillas_sf$boundary,
    cellsize = 0.5,
    n_group = 3,
    resolution = c(95, 80)
  )
  cvpart$block_ID <- seq_len(nrow(cvpart))
  cvpart$group <- NULL

  nests <- gorillas_sf$nests
  a <- sf::st_intersects(nests, cvpart)
  nests$.block <- unlist(a)

  fit1 <- lgcp(
    geometry ~ Intercept(1),
    data = nests,
    samplers = cvpart,
    domain = list(geometry = gorillas_sf$mesh),
    control.gcpo = list(enable = TRUE, type.cv = "joint")
  )
  result <- bru_block_gcpo(fit1)
  result$blocks
  result$gcpo
  sum(log(result$gcpo))

  # Multiple models
  pcmatern <- INLA::inla.spde2.pcmatern(
    gorillas_sf$mesh,
    prior.sigma = c(1, 0.01),
    prior.range = c(0.1, 0.01)
  )
  fit2 <- lgcp(
    geometry ~ Intercept(1) + field(geometry, model = pcmatern),
    data = nests,
    samplers = cvpart,
    domain = list(geometry = gorillas_sf$mesh),
    control.gcpo = list(enable = TRUE, type.cv = "joint")
  )
  result <- bru_block_gcpo(list(Model1 = fit1, Model2 = fit2))
  str(result)
}
```

---

bru_comp *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 `bru_comp` methods all rely on the `bru_comp.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 `bru_comp_list.formula()` method, called inside `bru()`.

Usage

```
bru_comp(...)

bru_component(...)

## S3 method for class 'character'
bru_comp(
  object,
  main = NULL,
  weights = NULL,
  ...,
  model = NULL,
  mapper = NULL,
  main_layer = NULL,
  main_selector = NULL,
  n = NULL,
  values = NULL,
  season.length = NULL,
  nrow = NULL,
  ncol = NULL,
  copy = NULL,
  weights_layer = NULL,
  weights_selector = NULL,
  group = 1L,
  group_mapper = NULL,
  group_layer = NULL,
  group_selector = NULL,
  ngroup = NULL,
  control.group = NULL,
  replicate = 1L,
  replicate_mapper = NULL,
  replicate_layer = NULL,
  replicate_selector = NULL,
  nrep = NULL,
```

```
  marginal = NULL,
  A.msk = deprecated(),
  .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()`. `main` supports `tidy` evaluation expressions, with `.data` and `.env` pronouns.
`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. Models with matrix input for `Cmatrix` or `graph` will use the matrix size. An error is given if it realises 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)
`nrow`, `ncol`	Number of rows and columns for `model` types "rw2d", "rw2diid", and "matern2d". Default is `NULL`.

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
marginal	May specify a bm_marginal() mapper, that is applied before scaling by weights.
A.msk	**[Deprecated]** and has no effect.
.envir	Evaluation environment. Can later be accessed via bru_comp_env(). Default: parent.frame()
envir_extra	Environment for storing INLA::f() argument values. If NULL, an new environment is created. Can be accessed via bru_comp_env_extra()

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 η would be written as

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

where:

c is the *intercept*

x is a *covariate*

β is a *latent variable* associated with x and

$\psi = \beta x$ 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 $\psi = \beta * x$ as x. Hence, it is not clear when x refers to the covariate or the effect of the covariate.

The bru_comp.character method is inlabru's equivalent to INLA's f() function but adds functionality that is unique to inlabru.

The main, weights, group, replicate, and *_layer arguments support tidy evaluation expressions, with .data and .env pronouns.

Functions

- bru_component(): Backwards compatibility alias for bru_comp()

Naming random effects

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

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

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

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

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

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

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

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

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

which corresponds to the mathematical expression e^{β}.

Author(s)

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

See Also

`bru_input()`, `summary.bru_comp()`

Other component constructors: `bru_comp_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 <- bru_comp_list(~ myLinearEffectOfX(main = x, model = "linear"))
summary(cmp)
# Equivalent shortcuts:
cmp <- bru_comp_list(~ myLinearEffectOfX(x, model = "linear"))
cmp <- bru_comp_list(~ myLinearEffectOfX(x))
# Individual component
cmp <- bru_comp("myLinearEffectOfX", main = x, model = "linear")
```

```
summary(cmp)

if (bru_safe_inla()) {
  # 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 <- bru_comp("myEffectOfX", main = x, model = "linear")
  summary(cmp)

  # A more complicated component:
  cmp <- bru_comp("myEffectOfX",
    main = x,
    model = INLA::inla.spde2.matern(fmesher::fm_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 <- bru_comp("eff", ~ -1 + x:z, model = "fixed")
  summary(cmp)
}
```

`bru_comp_env_extra` *Get/set component environment data*

Description

Get or set data in the component's `env_extra` and `env` environments. If `name` is `NULL`, the entire environment is returned.

Usage

```
bru_comp_env_extra(x, name = NULL)

bru_comp_env_extra(x, name) <- value

bru_comp_env(x, name = NULL)

bru_comp_env(x, name) <- value
```

Arguments

`x`	A `bru_comp` object
`name`	The name of the variable to get or set. Names prefixed by `"BRU_"` are reserved for internal use and should not be set by users or external package authors.
`value`	The value to set for `name` in `env_extra` (for the setter function).

Value

For the getter, either the entire environment or the value of `name` in the environment. For the setters, the modified `bru_comp` object.

Functions

- `bru_comp_env_extra(x, name) <- value`: Set data in the component's `env_extra` environment
- `bru_comp_env()`: Get the component's `env` environment, or an element from that environment. Note that in most cases this environment is the global R environment.
- `bru_comp_env(x, name) <- value`: Set data in the component's `env` environment. Note that in most cases this environment is the global R environment, so modifying it is normally not recommended.

See Also

bru_comp

Examples

```
if (bru_safe_inla()) {
  cmp <- bru_comp("x", x)
  as.list(bru_comp_env_extra(cmp))

  cmp <- bru_comp("x", x, model = "fixed")
  as.list(bru_comp_env_extra(cmp))

  bru_comp_env(cmp)
}
```

---

bru_comp_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

```
bru_comp_eval(
  main,
  group = NULL,
  replicate = NULL,
  weights = NULL,
  .state = NULL
)
```

Arguments

`main`, `group`, `replicate`, `weights`

Specification of where to evaluate a component. The four inputs are passed on to the joint `bru_mapper` for the component, as

```
list(mapper = list(
       main = main,
       group = group,
       replicate = replicate),
     scale = weights)
```

NOTE: If you have model component with the same name as a data variable you want to supply as input to `name_eval()`, you need to use `.data.[["myvar"]]` to access it. Otherwise, it will try to use the other component effect as input, which is ill-defined.

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

Value

A vector of values for a component

Examples

```
if (bru_safe_inla() &&
  require("sf", quietly = TRUE) &&
  requireNamespace("sn", quietly = TRUE)) {
  mesh <- fmesher::fm_mesh_2d_inla(
    cbind(0, 0),
    offset = 2,
    max.edge = 2.5
  )
  spde <- INLA::inla.spde2.pcmatern(
    mesh,
    prior.range = c(1, NA),
    prior.sigma = c(0.2, NA)
  )
  set.seed(12345L)
  data <- sf::st_as_sf(
    data.frame(
      x = runif(50),
      y = runif(50),
      z = rnorm(50)
    ),
    coords = c("x", "y")
  )
  fit <- bru(
    z ~ -1 + field(geometry, model = spde),
    family = "gaussian", data = data,
    options = list(control.inla = list(int.strategy = "eb"))
  )
```

```
  pred <- generate(
    fit,
    newdata = data.frame(A = 0.5, B = 0.5),
    formula = ~ field_eval(cbind(A, B)),
    n.samples = 1L
  )
}
```

---

bru_comp_list *Methods for inlabru component lists*

Description

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

Usage

```
bru_comp_list(object, ..., .envir = parent.frame())

## S3 method for class 'formula'
bru_comp_list(object, ..., lhoods = NULL, .envir = parent.frame())

## S3 method for class 'list'
bru_comp_list(
  object,
  ...,
  lhoods = NULL,
  .envir = parent.frame(),
  inputs = NULL
)

## S3 method for class 'bru_comp'
bru_comp_list(object, ..., .envir = parent.frame())

## S3 method for class 'bru_comp_list'
bru_comp_list(object, ..., .envir = parent.frame())

## S3 method for class 'bru_comp_list'
c(...)

## S3 method for class 'bru_comp'
c(...)

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

Arguments

object	The object to operate on
...	Parameters passed on to other methods. Also see Details.
.envir	An evaluation environment for non-formula input
lhoods	A bru_obs_list object
inputs	A tree-like list of component input evaluations, from bru_input.bru_obs_list().
x	bru_comp_list object from which to extract a sub-list
i	indices specifying elements to extract

Methods (by class)

- bru_comp_list(formula): Convert a component formula into a bru_comp_list object
- bru_comp_list(list): Combine a list of components, component lists, and/or component formulas into a single bru_comp_list object
- bru_comp_list(bru_comp): Place a single bru_comp object into a bru_comp_list object.
- bru_comp_list(bru_comp_list): Make sure a bru_comp_list object is fully configured.

Methods (by generic)

- c(bru_comp_list): The ... arguments should be bru_comp_list objects. The environment from the first argument will be applied to the resulting bru_comp_list.

Functions

- c(bru_comp): The ... arguments should be component objects from bru_comp(). The environment from the first argument will be applied to the resulting bru_comp_list.

Author(s)

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

See Also

Other component constructors: bru_comp()

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

`bru_convergence_plot` *Plot inlabru convergence diagnostics*

Description

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

Usage

```
bru_convergence_plot(x, from = 1, to = NULL, type = NULL)
```

Arguments

`x`	a bru object, typically a result from `bru()` for a nonlinear predictor model
`from, to`	integer values for the range of iterations to plot. Default `from = 1` (start from the first iteration) and `to = NULL` (end at the last iteration). Set `from = 0` to include the initial linearisation point in the track plot.
`type`	**[Experimental]** character; "bru" (default) for iterative nonlinear inlabru convergence diagnostics plots, or "inla" for INLA optimiser trace plots.

Details

Requires the "dplyr", "ggplot2", and "patchwork" packages to be installed.

Value

A ggplot object with four panels of convergence diagnostics:

- `Tracks`: Mode and linearisation values for each effect
- `Mode - Lin`: Difference between mode and linearisation values for each effect
- `|Change| / sd`: Absolute change in mode and linearisation values divided by the standard deviation for each effect
- `Change & sd`: Absolute change in mode and linearisation values and standard deviation for each effect

For multidimensional components, only the overall average, maximum, and minimum values are shown.

See Also

`bru()`

Examples

```
## 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

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

Arguments

`data`	A SpatialPointsDataFrame, SpatialPixelsDataFrame, SpatialGridDataFrame, SpatRaster, 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 `bru_comp()` for details.
`batch_size`	**[Deprecated]** due to improved algorithm. Size of nearest-neighbour calculation blocks, to limit the memory and computational complexity.

Value

An infilled vector of values

Examples

```
## Not run:
if (require("sf", quietly = TRUE)) {
  points <-
    sf::st_as_sf(
      data.frame(
        x = 1:3,
        y = 4:6,
        val = c(NA, NA, NA)
      ),
      coords = c("x", "y")
    )
```

```
  input_coord <- expand.grid(x = 0:7, y = 0:7)
  input <-
    sf::st_as_sf(
      cbind(input_coord, val = as.vector(input_coord$y)),
      coords = c("x", "y")
    )
  points$val <- bru_fill_missing(input, points, points$val)
  print(points)

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

## End(Not run)
```

`bru_gcpo_table` *Compare block-averaged GCPO scores across multiple fitted bru models*

Description

Calls `bru_block_gcpo()` on each fit and combines the results into a `data.frame` with one row per block and one column per model, making it straightforward to compare GCPO scores across models block by block.

Usage

```
bru_gcpo_table(fits = NULL, ...)
```

Arguments

`fits`	A named list of fitted `bru` objects, or `NULL` if models are passed via `\dots`.
`...`	Named fitted `bru` objects, used when `fits` is not a list.

Value

For a single-likelihood model, a `data.frame` with columns:

`block` Block labels from `response_data$BRU_block` (for `"cp"` models) or observation indices (for other models).

one column per model GCPO scores on the probability scale, named after the elements of `fits` or `\dots`.

For multi-likelihood models, a named list of such `data.frames`, one per likelihood.

See Also

bru_block_gcpo()

Examples

```
if (bru_safe_inla()) {
  cvpart <- cv_hex(
    gorillas_sf$boundary,
    cellsize = 0.5,
    n_group = 3,
    resolution = c(95, 80)
  )
  cvpart$block_ID <- seq_len(nrow(cvpart))
  cvpart$group <- NULL

  nests <- gorillas_sf$nests
  a <- sf::st_intersects(nests, cvpart)
  nests$.block <- unlist(a)

  fit1 <- lgcp(
    geometry ~ Intercept(1),
    data = nests,
    samplers = cvpart,
    domain = list(geometry = gorillas_sf$mesh),
    control.gcpo = list(enable = TRUE, type.cv = "joint")
  )
  pcmatern <- INLA::inla.spde2.pcmatern(
    gorillas_sf$mesh,
    prior.sigma = c(1, 0.01),
    prior.range = c(0.1, 0.01)
  )
  fit2 <- lgcp(
    geometry ~ Intercept(1) + field(geometry, model = pcmatern),
    data = nests,
    samplers = cvpart,
    domain = list(geometry = gorillas_sf$mesh),
    control.gcpo = list(enable = TRUE, type.cv = "joint")
  )
  gcpo_df <- bru_gcpo_table(Model1 = fit1, Model2 = fit2)
  names(gcpo_df)
  colSums(log(gcpo_df[, -1]))
}
```

---

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, ...)

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

bru_get_mapper_safely(model, ...)

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

Arguments

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

Details

Before implementing your own `bru_get_mapper` method, check if there is already a general method available that handles your model class, such as `bru_get_mapper.inla.spde()`, `bru_get_mapper.inla.rgeneric()`, and `bru_get_mapper.inla.cgeneric()`.

Value

A bru_mapper object defined by the model component

Methods (by class)

- `bru_get_mapper(inla.spde)`: Extract an indexed mapper for the `model$mesh` object contained in the model object, which is assumed to be of a class supporting relevant `fmesher` methods.
- `bru_get_mapper(inla.rgeneric)`: Returns the mapper given by a pre-computed mapper, or an index mapper mapping the size of the model graph.

  The easiest method to define a mapper for an `inla.rgeneric` model is to store the mapper in the object. Alternatively, define your model using a subclass and define a `bru_get_mapper.<subclass>` method that should return the corresponding `bru_mapper` object.

The order of precedence for the mapper construction when calling `bru_get_mapper(model)` has the following precedence:

1. `bru_get_mapper.<subclass>`, if `model` has a subclass, otherwise
2. `model[["mapper"]]` if that is `NULL`, and otherwise
3. `bm_index()` using the size of the graph returned by `model[["f"]][["n"]]`

- `bru_get_mapper(inla.cgeneric)`: Works the same as the method of `inla.rgeneric`
- `bru_get_mapper(inla_model_reparam)`: Reparameterised inla model mapper, see `inla.spde2.pcmatern_B()` **[Experimental]**

Functions

- `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.

See Also

bru_mapper for mapper constructor methods, and the individual mappers for specific implementation details.

Other mappers: `bm_aggregate()`, `bm_collect()`, `bm_const()`, `bm_factor()`, `bm_fm_mesh_1d`, `bm_fmesher()`, `bm_harmonics()`, `bm_index()`, `bm_linear()`, `bm_logitaverage()`, `bm_logsumexp()`, `bm_marginal()`, `bm_matrix()`, `bm_multi()`, `bm_pipe()`, `bm_reparam()`, `bm_repeat()`, `bm_scale()`, `bm_shift()`, `bm_sum()`, `bm_taylor()`, `bru_mapper()`

Examples

```
if (bru_safe_inla()) {
  library(INLA)
  mesh <- fmesher::fm_rcdt_2d_inla(globe = 2)
  spde <- inla.spde2.pcmatern(mesh,
    prior.range = c(1, 0.5),
    prior.sigma = c(1, 0.5)
  )
  mapper <- bru_get_mapper(spde)
  ibm_n(mapper)
}
```

`bru_index` *Extract predictor or component index information*

Description

[Experimental] Extract the index vector for a `bru_obs()` predictor, or the whole or a subset of a full `bru()` predictor.

Usage

```
bru_index(object, ...)

## S3 method for class 'bru_obs'
bru_index(object, what = NULL, ...)

## S3 method for class 'bru_obs_list'
bru_index(object, tag = NULL, what = NULL, ...)

## S3 method for class 'bru'
bru_index(object, tag = NULL, what = NULL, ...)

## S3 method for class 'bru_info'
bru_index(object, tag = NULL, what = NULL, ...)

## S3 method for class 'bru_comp'
bru_index(object, inla_f, ...)

## S3 method for class 'bru_comp_list'
bru_index(object, inla_f, ...)

## S3 method for class 'bru_model'
bru_index(object, used, ...)

index_eval(...)
```

Arguments

object	A component.
...	Arguments passed on to sub-methods.
what	character or NULL; One of NULL, "all", "observed", and "missing". If NULL (default) or "all", gives the index vector for the full sub-model predictor. If "observed", gives the index vector for the observed part (response is not NA). If "missing", gives the index vector for the missing part (response is NA) of the model.
tag	character or integer; Either a character vector identifying the tags of one or more of the bru_obs() observation models, or an integer vector identifying models by their bru() specification order. If NULL (default) computes indices for all sub-models.
inla_f	logical; when TRUE, must result in values compatible with INLA::f(...) an specification and corresponding INLA::inla.stack(...) constructions.
used	A bru_used() object

Value

- bru_index(bru_obs): An integer vector.

- `bru_index(bru_obs_list)`: An `integer` vector.
- `bru_index(bru)`: An `integer` vector.
- `bru_index(bru_info)`: An `integer` vector.
- `bru_index(bru_comp)`: A list of indices into the latent variables compatible with the component mapper.
- `bru_index(bru_comp_list)`: A list of list of indices into the latent variables compatible with each component mapper.
- `bru_index(bru_model)`: 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.

Methods (by class)

- `bru_index(bru_obs)`: Extract the index vector for the predictor vector for a `bru_obs()` sub-model. The indices are relative to the sub-model, and need to be appropriately offset to be used in the full model predictor.
- `bru_index(bru_obs_list)`: Extract the index vector for "APredictor" for one or more specified observation `bru_obs()` sub-models. Accepts any combination of `tag` and `what`.
- `bru_index(bru)`: Extract the index vector for "APredictor" for one or more specified observation `bru_obs()` sub-models. Accepts any combination of `tag` and `what`.
- `bru_index(bru_info)`: Extract the index vector for "APredictor" for one or more specified observation `bru_obs()` sub-models. Accepts any combination of `tag` and `what`.
- `bru_index(bru_model)`: Compute all index values for a `bru_model()` object. Computes the index value matrices for included components according to the `used` argument.

Functions

- `index_eval()`: **[Deprecated]** since "2.12.0.9023". Use `bru_index()` instead.

Author(s)

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

Examples

```
fit <- bru(
  ~ 0 + x,
  bru_obs(
    y ~ .,
    data = data.frame(x = 1:3, y = 1:3 + rnorm(3)),
    tag = "A"
  ),
  bru_obs(
    y ~ .,
    data = data.frame(x = 1:4, y = c(NA, NA, 3:4) + rnorm(4)),
```

```
    tag = "B"
  ),
  options = list(bru_run = FALSE) # We only need the model structure
)
bru_index(fit)
bru_index(fit, "A")
bru_index(fit, "B")
bru_index(fit, c("B", "A"))
bru_index(fit, what = "missing")
```

---

bru_info *Methods for bru_info objects*

Description

The bru_info class is used to store metadata about bru models.

Usage

```
bru_info(...)

## S3 method for class 'character'
bru_info(method, ..., inlabru_version = NULL, INLA_version = NULL)

## S3 method for class 'bru'
bru_info(object, ...)

as_bru_info(object, ...)

## S3 method for class 'bru_info'
as_bru_info(object, ...)

## S3 method for class 'bru'
as_bru_info(object, ...)

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

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

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

Arguments

...	Additional arguments to be stored in the bru_info object. For summary and print methods, arguments passed on to submethods.
method	character; The type of estimation method used
inlabru_version	character; inlabru package version. Default: NULL, for automatically detecting the version
INLA_version	character; INLA package version. Default: NULL, for automatically detecting the version
object	A bru_info object
verbose	logical; If TRUE, include more details of the component definitions. If FALSE, only show basic component definition information. Default: FALSE
x	An object to be printed

Methods (by class)

- bru_info(character): Create a bru_info object
- bru_info(bru): Extract the bru_info object from an estimated bru() result object. The default print method shows information about model components and observation models.

Methods (by generic)

- as_bru_info(bru_info): Extract a bru_info object.

Functions

- as_bru_info(): Extract the bru_info object from an estimated bru() result object. The default print method shows information about model components and observation models.
- as_bru_info(bru): Extract the bru_info object from an estimated bru() result object.

---

bru_input *Store and evaluate component inputs*

Description

Store and evaluate component inputs, including support for non-standard-evaluation with rlang, automatic transfer to function calls, eval_spatial(), and MatrixModels::model.Matrix().

These functions are normally only called internally, but users may find it useful to call them directly to experiment, and to make advanced model definitions using ibm_input_set/ibm_input_new().

Usage

```
new_bru_input(input, label = NULL, layer = NULL, selector = NULL, ...)

bru_input(...)

## S3 method for class 'bru_input'
bru_input(
  x,
  data = NULL,
  mask = NULL,
  null.on.fail = FALSE,
  .envir = parent.frame(),
  ...
)

## S3 method for class 'bru_comp'
bru_input(x, ..., label = x$label)

## S3 method for class 'bru_comp_list'
bru_input(x, ...)

## S3 method for class 'bru_model'
bru_input(x, lhoods, ...)

## S3 method for class 'bru_obs'
bru_input(x, components, ...)

## S3 method for class 'bru_obs_list'
bru_input(x, components, ...)

## S3 method for class 'bru_mapper'
bru_input(x, ..., label = "<unknown>")

## S3 method for class 'bm_pipe'
bru_input(x, ..., label = "<unknown>")

## S3 method for class 'bm_multi'
bru_input(x, ..., label = "<unknown>")

## S3 method for class 'bm_collect'
bru_input(x, ..., label = "<unknown>")

## S3 method for class 'bm_repeat'
bru_input(x, ..., label = "<unknown>")

## S3 method for class 'bm_sum'
bru_input(x, ..., label = "<unknown>")
```

Arguments

input	An expression to be evaluated.
label	character; optional label used to identify the object in informational messages.
layer	Optional expression that evaluates layer names or indices for use with eval_spatial()
selector	character or integer; optional selector for use with use with eval_spatial()
...	Passed on to sub-methods.
x	A bru_input object, or other object for recursive evaluation.
data	A data.frame, tibble, sf, list, or Spatial* object of covariates and/or point locations.
mask	A bru_data_mask object, constructed internally.
null.on.fail	logical; if TRUE, return NULL if the input evaluation fails. If FALSE (default), return a vector of 1s and issue a warning (only for deprecated use of coordinates without having loaded the sp package), or stop with an error.
.envir	environment in which to evaluate the input expression. Default is parent.frame()
lhoods	A bru_obs_list object containing all observations models.
components	A bru_comp_list object containing all components defined in the model.

Value

- bru_input(bru_comp): A list of mapper input values, formatted for the full component mapper (of type bm_pipe)
- bru_input(bru_comp_list): A list of mapper input values, with one entry for each component.
- bru_input(bru_model): A list of mapper input values, with one entry for each observation model, each containing a list of inputs for the components used by the corresponding observation model.
- bru_input(bru_obs): A list of mapper input values, for each of the components used by the corresponding observation model.
- bru_input(bru_obs_list): A list of mapper input values, with one entry for each observation model, each containing a list of inputs for the components used by the corresponding observation model.

Methods (by class)

- bru_input(bru_input): Attempts to evaluate a component input (e.g. main, group, replicate, or weight), and process the results:
  1. If rlang::eval_tidy() failed, return NULL or map everything to 1 (see the null.on.fail argument). This should normally not happen, unless the component use logic is incorrect, (via used) leading to missing columns for a certain likelihood in a multi-bru_obs() model.
  2. If we obtain a function, apply the function to the data object

3. If we obtain an object supported by `eval_spatial()`, extract the values of that data frame at the point locations
4. If we obtain a formula, call `ModelMatrix::model.Matrix()`
5. Else we obtain a vector and return as-is. This happens when input references a column of the data points, or some other complete expression

- `bru_input(bru_model)`: Computes the component inputs for included components for each observation model.
- `bru_input(bru_mapper)`: Evaluate the input associated with a `bru_mapper`.
- `bru_input(bm_pipe)`: Evaluate the inputs for each sub-mapper in a `bm_pipe` object.
- `bru_input(bm_multi)`: Evaluate the inputs for each sub-mapper in a `bm_multi` object.
- `bru_input(bm_collect)`: Evaluate the inputs for each sub-mapper in a `bm_collect` object.
- `bru_input(bm_repeat)`: Evaluate the inputs for the sub-mapper in a `bm_repeat` object.
- `bru_input(bm_sum)`: Evaluate the inputs for each sub-mapper in a `bm_sum` object.

Functions

- `new_bru_input()`: Create a `bru_input` object.
- `bru_input()`: Evaluate inputs

Simple covariates and the input parameters

It is not unusual for a random effect act on a transformation of a covariate. In other frameworks this would mean that the transformed covariate would have to be calculated in advance and added to the data frame that is usually provided via the `data` parameter. inlabru provides the option to do this transformation automatically. For instance, one might be interested in the effect of a covariate x^2. In inla and other frameworks this would require to add a column `xsquared` to the input data frame and use the formula

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

In inlabru this can be achieved in several ways of using the `main` parameter (`map` in version 2.1.13 and earlier), which does not need to be named.

- `components = y ~ psi(x^2, model = "linear")`
- `components = y ~ psi(main = x^2, model = "linear")`
- `components = y ~ psi(mySquareFun(x), model = "linear")`,
- `components = y ~ psi(myOtherSquareFun, model = "linear")`,

In the first example inlabru will interpret the map parameter as an expression to be evaluated within the data provided. Since x is a known covariate it will know how to calculate it. The second example is an expression as well but it uses a function called `mySquareFun`. This function is defined by user but has to be accessible within the work space when setting up the components. The third example provides the function `myOtherSquareFun`. In this case, inlabru will call the function as `myOtherSquareFun(.data.)`, where `.data.` is the data provided via the `bru_obs()` `data` parameter. The function needs to know what parts of the data to use to construct the needed output. For example,

```
myOtherSquareFun <- function(data) {
  data[ ,"x"]^2
}
```

Interactions can be handled by a formula input and `model = "fixed"`: `components = y ~ 0 + name(~ 1 + x:z, model = "fixed")`

Spatial Covariates

When fitting spatial models it is common to work with covariates that depend on space, e.g. sea surface temperature or elevation. Although it is straightforward to add this data to the input data frame or write a covariate function like in the previous section there is an even more convenient way in inlabru. Spatial covariates are often stored as `SpatialPixelsDataFrame`, `SpatialPixelsDataFrame` or `RasterLayer` objects. These can be provided directly via the input expressions if they are supported by `eval_spatial()`, and the `bru_obs()` data is an `sf` or `SpatialPointsDataFrame` object. `inlabru` will then automatically evaluate and/or interpolate the covariate at your data locations when using code like

```
components = y ~ psi(mySpatialPixels, model = "linear")
```

For more precise control, use the the `layer` and `selector` arguments (see `bru_comp()`), or call `eval_spatial()` directly, e.g.:

```
components = y ~ psi(eval_spatial(mySpatialPixels, where = .data.),
                     model = "linear")
```

Coordinates

A common spatial modelling component when using inla are SPDE models. An important feature of inlabru is that it will automatically calculate the so called A-matrix (a component model matrix) which maps SPDE values at the mesh vertices to values at the data locations. For this purpose, the input can be set to `coordinates`, which is the `sp` package function that extracts point coordinates from the `SpatialPointsDataFrame` that was provided as input to `bru_obs()`. The code for this would look as follows:

```
components = y ~ field(coordinates, model = inla.spde2.matern(...))
```

Since `coordinates` is a function from the `sp` package, this results in evaluation of `sp::coordinates(.data.)`, which loses any CRS information from the data object.

For `sf` data with a geometry column (by default named `geometry`), use

```
components = y ~ field(geometry, model = inla.spde2.matern(...))
```

Since the CRS information is part of the geometry column of the `sf` object, this retains CRS information, so this is more robust, and allows the model to be built on a different CRS than the observation data.

Author(s)

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

See Also

bru_input_text()

summary.bru_input(), bru_comp()

ibm_input

Examples

```
(inp <- new_bru_input(x, "LABEL"))
bru_input(inp, data.frame(x = 1:3))
```

---

bru_is_additive *Check for predictor expression additivity*

Description

Checks if a predictor expression is additive or not

Usage

```
bru_is_additive(x, ...)

## S3 method for class 'character'
bru_is_additive(x, ..., verbose = FALSE)

## S3 method for class 'expression'
bru_is_additive(x, ..., verbose = FALSE)

## S3 method for class 'formula'
bru_is_additive(x, ..., verbose = FALSE)

## S3 method for class 'bru_pred_expr'
bru_is_additive(x, ...)

## S3 method for class 'bru_obs'
bru_is_additive(x, ...)

## S3 method for class 'bru_obs_list'
bru_is_additive(x, ...)
```

Arguments

x	A predictor expression, formula, or parse information data.frame.
...	Arguments passed on recursively.
verbose	logical; if TRUE, print diagnostic parsing information.

Value

TRUE if the expression is detected to be additive, FALSE otherwise.

Examples

```
bru_is_additive(~ x + y)
bru_is_additive(~ x * y)
```

---

bru_is_linear *Check for predictor linearity*

Description

Checks if a predictor expression is linear (or affine)

Usage

```
bru_is_linear(x, ...)

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

## S3 method for class 'bru_info'
bru_is_linear(x, ...)

## S3 method for class 'bru_model'
bru_is_linear(x, ...)

## S3 method for class 'bru_obs'
bru_is_linear(x, ...)

## S3 method for class 'bru_obs_list'
bru_is_linear(x, ...)

## S3 method for class 'bru_pred_expr'
bru_is_linear(x, ...)

## S3 method for class 'bru_comp_list'
bru_is_linear(x, ...)
```

```
## S3 method for class 'bru_comp'
bru_is_linear(x, ...)

## S3 method for class 'bru_mapper'
bru_is_linear(x, ...)
```

Arguments

x	A object containing a predictor definition
...	Arguments passed on recursively.

Value

TRUE if the expression is detected to be linear, FALSE otherwise.

Examples

```
bru_is_linear(new_bru_pred_expr(~ x + y))
bru_is_linear(new_bru_pred_expr(~ x * y))
bru_is_linear(bm_scale())
bru_is_linear(bm_logsumexp())
```

---

bru_is_rowwise *Check for predictor rowwise evaluability*

Description

Checks if a predictor expression may be evaluated rowwise

Usage

```
bru_is_rowwise(x, ...)

## S3 method for class 'bru_pred_expr'
bru_is_rowwise(x, ...)

## S3 method for class 'bru_obs'
bru_is_rowwise(x, ...)

## S3 method for class 'bru_obs_list'
bru_is_rowwise(x, ...)

## S3 method for class 'bru_comp_list'
bru_is_rowwise(x, ...)

## S3 method for class 'bru_comp'
```

```
bru_is_rowwise(x, ...)

## S3 method for class 'bru_mapper'
bru_is_rowwise(x, ...)
```

Arguments

x	An object containing a predictor definition
...	Arguments passed on to submethods.

Value

TRUE if the expression is believed to be rowwise, FALSE otherwise.

Examples

```
bru_is_rowwise(new_bru_pred_expr(~ x + y))
```

---

bru_log *Access methods for* bru_log *objects*

Description

Access method for bru_log objects. Note: Up to version 2.8.0, bru_log() was a deprecated alias for bru_log_message(). When running on 2.8.0 or earlier, use bru_log_get() to access the global log, and cat(fitbru_iinlalog, sep = "\n") to print a stored estimation object log. After version 2.8.0, use bru_log() to access the global log, and bru_log(fit) to access a stored estimation log.

Usage

```
bru_log(x = NULL, verbosity = NULL)

## S3 method for class 'character'
bru_log(x, verbosity = NULL)

## S3 method for class 'bru_log'
bru_log(x, verbosity = NULL)

## S3 method for class 'iinla'
bru_log(x, verbosity = NULL)

## S3 method for class 'bru'
bru_log(x, verbosity = NULL)

## S3 method for class 'bru_log'
```

```
format(x, ..., timestamp = TRUE, verbosity = FALSE)

## S3 method for class 'bru_log'
print(x, ..., timestamp = TRUE, verbosity = FALSE)

## S3 method for class 'bru_log'
as.character(x, ...)

## S3 method for class 'bru_log'
x[i]

## S3 method for class 'bru_log'
c(...)

## S3 method for class 'bru_log'
length(x)
```

Arguments

x	An object that is, contains, or can be converted to, a bru_log object. If NULL, refers to the global inlabru log.
verbosity	integer value for limiting the highest verbosity level being returned.
...	further arguments passed to or from other methods.
timestamp	If TRUE, include the timestamp of each message. Default TRUE.
i	indices specifying elements to extract. If character, denotes the sequence between bookmark i and the next bookmark (or the end of the log if i is the last bookmark)

Value

bru_log A bru_log object, containing a character vector of log messages, and potentially a vector of bookmarks.

Methods (by generic)

- format(bru_log): Format a bru_log object for printing. If verbosity is TRUE, include the verbosity level of each message.
- print(bru_log): Print a bru_log object with cat(x, sep = "\n"). If verbosity is TRUE, include the verbosity level of each message.
- as.character(bru_log): Convert bru_log object to a plain character vector
- [: Extract a subset of a bru_log object
- c(bru_log): Concatenate several bru_log or character objects into a bru_log object.
- length(bru_log): Obtain the number of log entries into a bru_log object.

Functions

- bru_log(): Extract stored log messages. If non-NULL, the verbosity argument determines the maximum verbosity level of the messages to extract.

See Also

Other inlabru log methods: bru_log_bookmark(), bru_log_message(), bru_log_new(), bru_log_offset(), bru_log_reset()

Examples

```
bru_log(verbosity = 2L)
format(bru_log())

bru_log(verbosity = 2L)
print(bru_log(), timestamp = TRUE, verbosity = TRUE)
```

---

bru_log_bookmark *Methods for* bru_log *bookmarks*

Description

Methods for bru_log bookmarks.

Usage

```
bru_log_bookmark(bookmark = "", offset = NULL, x = NULL)

bru_log_bookmarks(x = NULL)
```

Arguments

bookmark	character; The label for a bookmark with a stored offset.
offset	integer; a position offset in the log, with 0L pointing at the start of the log. If negative, denotes the point abs(offset) elements from tail of the log. When bookmark is non-NULL, the offset applies a shift (forwards or backwards) to the bookmark list.
x	A bru_log object. If NULL, the global inlabru log is used.

Value

bru_log_bookmark(): Returns the modified bru_log object if x is non-NULL.

bru_log_bookmarks(): Returns the bookmark vector associated with x

Functions

- bru_log_bookmark(): Set a log bookmark. If offset is NULL (the default), the bookmark will point to the current end of the log.
- bru_log_bookmarks(): Return an integer vector with named elements being bookmarks into the global inlabru log with associated log position offsets.

See Also

Other inlabru log methods: bru_log(), bru_log_message(), bru_log_new(), bru_log_offset(), bru_log_reset()

---

bru_log_message *Add a log message*

Description

Adds a log message.

Usage

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

bru_log_abort(
  msg,
  ...,
  domain = NULL,
  appendLF = TRUE,
  verbosity = 1L,
  allow_verbose = TRUE,
  verbose = FALSE,
  verbose_store = NULL,
  call = rlang::caller_env(),
  .frame = rlang::caller_env()
)

bru_log_warn(
  msg,
  ...,
  domain = NULL,
  appendLF = TRUE,
  verbosity = 1L,
  allow_verbose = TRUE,
  verbose = FALSE,
  verbose_store = NULL,
```

```
  call = rlang::caller_env(),
  .frame = rlang::caller_env()
)
```

Arguments

...	For bru_log_message(), zero or more objects passed on to base::.makeMessage(). For bru_log_abort() and bru_log_warn(), passed on to rlang::abort() and rlang::warn().
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, numeric, or NULL; local override for verbose output. If NULL, the global option bru_verbose or default value is used. If FALSE, no messages are printed. If TRUE, messages with verbosity ≤ 1 are printed. If numeric, messages with verbosity $\leq$ verbose are printed.
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().
x	A bru_log object. If NULL, refers to the global inlabru log.
msg	character; passed to base::.makeMessage()
call	The calling environment.
.frame	The throwing context, for when .internal is TRUE

Value

bru_log_message returns invisible(x), where x is the updated bru_log object, or NULL.

Functions

- bru_log_abort(): Store a log message and throw an error.
- bru_log_warn(): Store a log message and throw a warning.

See Also

Other inlabru log methods: bru_log(), bru_log_bookmark(), bru_log_new(), bru_log_offset(), bru_log_reset()

Examples

```
if (interactive()) {
  code_runner <- function() {
    bru_options_set_local(
      # Show messages up to and including level 2 (default 0)
      bru_verbose = 2,
      # Store messages to an including level 3 (default Inf, storing all)
```

```
    bru_verbose_store = 3
  )

  bru_log_bookmark("bookmark 1")
  bru_log_message("Test message 1", verbosity = 1)
  bru_log_message("Test message 2", verbosity = 2)
  bru_log_bookmark("bookmark 2")
  bru_log_message("Test message 3", verbosity = 3)
  bru_log_message("Test message 4", verbosity = 4)

  invisible()
}
message("Run code")
code_runner()
message("Check log from bookmark 1")
print(bru_log()["bookmark 1"])
message("Check log from bookmark 2")
print(bru_log()["bookmark 2"])
}
```

`bru_log_new` *Create a* `bru_log` *object*

Description

Create a `bru_log` object, by default empty.

Usage

```
bru_log_new(x = NULL, bookmarks = NULL)
```

Arguments

`x`	An optional character vector of log messages, or `data.frame` with columns `message`, `timestamp`, and `verbosity`.
`bookmarks`	An optional `integer` vector of named bookmarks message in `x`.

See Also

Other inlabru log methods: `bru_log()`, `bru_log_bookmark()`, `bru_log_message()`, `bru_log_offset()`, `bru_log_reset()`

Examples

```
x <- bru_log_new()
x <- bru_log_message("Test message", x = x)
print(x)
```

bru_log_offset *Position methods for* bru_log *objects*

Description

Position methods for bru_log objects.

Usage

```
bru_log_offset(x = NULL, bookmark = NULL, offset = NULL)

bru_log_index(x = NULL, i, verbosity = NULL)
```

Arguments

x	A bru_log object. If NULL, the global inlabru log is used.
bookmark	character; The label for a bookmark with a stored offset.
offset	integer; a position offset in the log, with 0L pointing at the start of the log. If negative, denotes the point abs(offset) elements from tail of the log. When bookmark is non-NULL, the offset applies a shift (forwards or backwards) to the bookmark list.
i	indices specifying elements to extract. If character, denotes the sequence between bookmark i and the next bookmark (or the end of the log if i is the last bookmark)
verbosity	integer value for limiting the highest verbosity level being returned.

Functions

- bru_log_offset(): Utility function for computing log position offsets.
- bru_log_index(): Utility function for computing index vectors for bru_log objects.

See Also

Other inlabru log methods: bru_log(), bru_log_bookmark(), bru_log_message(), bru_log_new(), bru_log_reset()

`bru_log_reset` *Clear log contents*

Description

Clears the log contents up to a given `offset` or `bookmark`. Default: clear the entire log. When `x` is NULL, the global `inlabru` log is updated, and `invisible(NULL)` is returned. Otherwise the updated object is returned (invisibly).

Usage

```
bru_log_reset(x = NULL, bookmark = NULL, offset = NULL)
```

Arguments

`x`	A `bru_log` object, or in some cases, and object that can be converted/extracted to a `bru_log` object. `NULL` denotes the global `inlabru` log object.
`bookmark`	character; The label for a bookmark with a stored offset.
`offset`	integer; a position offset in the log, with `0L` pointing at the start of the log. If negative, denotes the point `abs(offset)` elements from tail of the log. When `bookmark` is non-NULL, the `offset` applies a shift (forwards or backwards) to the bookmark list.

Value

Returns (invisibly) the modified `bru_log` object, or `NULL` (when `x` is `NULL`)

See Also

Other inlabru log methods: `bru_log()`, `bru_log_bookmark()`, `bru_log_message()`, `bru_log_new()`, `bru_log_offset()`

Examples

```
## Not run:
if (interactive()) {
  bru_log_reset()
}

## End(Not run)
```

bru_mapper *Constructors for* bru_mapper *objects*

Description

Constructors for bru_mapper objects

Usage

```
bru_mapper(...)

bru_mapper_define(mapper, new_class = NULL, remove_class = "list", ...)
```

Arguments

...	Arguments passed on to sub-methods, or used for special purposes, see details for each function below.
mapper	For bru_mapper_define, a prototype mapper object, see Details.
new_class	If non-NULL, this is added at the front of the class definition
remove_class	If non-NULL, this class or classes is removed from the class definition before adding the new_class names. Default is "list".

Value

A bru_mapper 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 mapper clesses in user code. To extracting mappers for latent component models, see bru_get_mapper().
- 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
  ```

  etc., which semi-automates it.

See Also

bru_mapper_generics for generic methods, the individual mapper pages for special method implementations, and bru_get_mapper for hooks to extract mappers from latent model object class objects.

Other mappers: bm_aggregate(), bm_collect(), bm_const(), bm_factor(), bm_fm_mesh_1d, bm_fmesher(), bm_harmonics(), bm_index(), bm_linear(), bm_logitaverage(), bm_logsumexp(), bm_marginal(), bm_matrix(), bm_multi(), bm_pipe(), bm_reparam(), bm_repeat(), bm_scale(), bm_shift(), bm_sum(), bm_taylor(), bru_get_mapper()

Examples

```
mapper <- bm_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 ibm_n() and ibm_values() are available. Otherwise the ibm_n() and ibm_values() methods also need to be provided.

See Also

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

Other mapper methods: ibm_eval(), ibm_eval2(), ibm_inla_subset(), ibm_invalid_output(), ibm_is_linear(), ibm_is_rowwise(), ibm_jacobian(), ibm_linear(), ibm_n(), ibm_n_output(), ibm_names(), ibm_simplify(), ibm_values()

---

bru_model_mapper_methods *Mapper methods for model objects*

Description

Methods for the ibm_linear() and ibm_simplify() methods for bru model objects and related classes.

Usage

```
## S3 method for class 'bru_model'
ibm_linear(mapper, input, state = NULL, ...)

## S3 method for class 'bru_comp_list'
ibm_linear(mapper, input, state = NULL, ...)

## S3 method for class 'bru_model'
ibm_simplify(mapper, input = NULL, state = NULL, ...)

## S3 method for class 'bru_comp'
ibm_simplify(mapper, input = NULL, state = NULL, ...)

## S3 method for class 'bru_comp_list'
ibm_simplify(mapper, input = NULL, state = NULL, ...)

## S3 method for class 'bm_list'
ibm_linear(mapper, input, state = NULL, ...)

## S3 method for class 'bm_list'
ibm_simplify(mapper, input = NULL, state = NULL, ...)
```

Arguments

mapper	A mapper S3 object, inheriting from bru_mapper.
input	Data input for the mapper.
state	A vector of latent state values for the mapping, of length ibm_n(mapper, inla_f = FALSE)
...	Arguments passed on to other methods

Functions

- ibm_linear(bru_model): Returns a list (one element per observation model) of bm_list objects, each with one bm_taylor entry for each included component.
- ibm_simplify(bru_model): Returns a list (one element per observation model) of bm_list objects, each with one bru_mapper entry for each included component.

---

bru_names *Extract standardised names from a bru or inla result object*

Description

Extracts the names of fixed effects, random effects, and hyperparameters, converted with bru_standardise_names()

Usage

```
bru_names(x)

## S3 method for class 'inla'
bru_names(x)

## S3 method for class 'bru'
bru_names(x)
```

Arguments

x	an inla or bru() result object

Value

A character vector with standardised names

Examples

```
if (bru_safe_inla()) {
  fit <- bru(y ~ 1 + x + z(z, model = "iid"),
    data = data.frame(
      y = rnorm(10),
      x = rnorm(10),
      z = rep(seq_len(2), 5)
    )
  )
  bru_names(fit)
}
```

---

bru_obs *Observation model construction for usage with bru()*

Description

Observation model construction for usage with bru().

Note: Prior to version 2.12.0, this function was called like(), and that alias will remain for a while until examples etc have been updated and users made aware of the change. The name change is to avoid issues with namespace clashes, e.g. with data.table::like(), and also to signal that the function defines observation models, not just likelihood functions.

Usage

```
bru_obs(
  formula = . ~ .,
  family = "gaussian",
  data = NULL,
```

```
  response_data = NULL,
  data_extra = NULL,
  E = NULL,
  Ntrials = NULL,
  weights = NULL,
  scale = NULL,
  domain = NULL,
  samplers = NULL,
  ips = NULL,
  used = NULL,
  is_rowwise = NULL,
  aggregate = NULL,
  aggregate_input = NULL,
  control.family = NULL,
  control.gcpo = NULL,
  tag = NULL,
  options = list(),
  .envir = parent.frame(),
  include = deprecated(),
  exclude = deprecated(),
  include_latent = deprecated(),
  allow_combine = deprecated()
)

like(
  ...,
  E = NULL,
  Ntrials = NULL,
  weights = NULL,
  scale = NULL,
  options = list(),
  .envir = parent.frame()
)

bru_obs_list(..., .tag = NULL)

## S3 method for class 'list'
bru_obs_list(object, ..., .tag = NULL)

## S3 method for class 'bru_obs'
bru_obs_list(..., .tag = NULL)

## S3 method for class 'bru_obs_list'
bru_obs_list(..., .tag = NULL)

## S3 method for class 'bru_obs'
c(...)
```

```
## S3 method for class 'bru_obs_list'
c(...)

## S3 method for class 'bru_obs_list'
x[i]

like_list(...)

bru_like_list(...)
```

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()$like inlabru supports fitting latent Gaussian Cox processes via family = "cp". As an alternative to bru(), the lgcp() function provides a convenient interface to fitting Cox processes.
data	Predictor expression-specific data, as a data.frame, tibble, or sf. Since 2.12.0.9023, deprecated support for SpatialPoints[DataFrame] objects.
response_data	Observation/response-specific data for models that need different size/format for inputs and response variables, as a data.frame, tibble, or sf. Since 2.12.0.9023, deprecated support for SpatialPoints[DataFrame] objects.
data_extra	object convertible with as.list() with additional variables to be made available in predictor evaluations. Variables with the same names as the data object will be ignored, unless accessed via .data_extra.[["name"]] or .data_extra.$name in the formula.
E	Exposure/effort parameter for family = 'poisson' passed on to INLA::inla. Special case if family is 'cp': rescale all integration weights by a scalar E. For sampler specific reweighting/effort, use a weight column in the samplers object instead, see fmesher::fm_int(). 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. For family = "cp", the weights are applied as sum(weights * eta) in the point location contribution part of the log-likelihood, where eta is the linear predictor, and do not affect the integration part of the likelihood. This can be used to implement approximative methods for point location uncertainty.
scale	Fixed (optional) scale parameters of the precision for several models, such as Gaussian and student-t response models.
domain, samplers, ips	Arguments used for family="cp" and aggregate=.

domain Named list of domain definitions, see fmesher::fm_int().

samplers Integration domain for family="cp" or subdomains for aggregate=, see fmesher::fm_int().

ips Integration points. Defaults to fmesher::fm_int(domain, samplers). If explicitly given, overrides domain and samplers. fmesher::new_fm_int() (from fmesher 0.5.0.9013) can be used for manually constructed integration schemes.

used Either NULL (default) or a bru_used() object. When, NULL, the information about what effects and latent vectors are made available to the predictor evaluation is defined by bru_used(formula), which will include all effects and latent vectors used by the predictor expression.

is_rowwise, allow_combine

logical; If is_rowwise is FALSE, the predictor expression may involve several rows of the input data to influence the same row. When NULL, it defaults to TRUE, unless response_data is non-NULL, or data is a list, or the likelihood construction requires it. The allow_combine argument is a deprecated and inverted version, such that is_rowwise = !allow_combine.

aggregate character ("none" or a valid name for the type argument of bm_aggregate()) or an aggregation bru_mapper object (bm_aggregate(), bm_logsumexp(), or bm_logitaverage()). Default NULL, interpreted as "none". **[Experimental]**, available from version 2.12.0.9013.

aggregate_input

NULL or an optional input list to the mapper defined by non-NULL aggregate, overriding the default,

```
list(block = .data.[[".block"]],
     weights = .data.[["weight"]],
     n_block = bru_response_size(.response_data.),
     block_response = ".block")
```

[Experimental], available from version 2.12.0.9013.

From 2.13.0.9016, it will look for a block_response element in the list, which should be the name of a response variable in response_data to match the block information against, allowing character or factor aggregation block information to be used by replacing block with match(block, block_response). If not supplied, the name ".block" is tried. If that isn't available, the block information must be supplied directly as a numeric or integer vector, indexing into the rows of the response variable. Having no block_response variable is equivalent to having response_data$.block = seq_len(bru_response_size(response_data)).

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

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

tag character; Name that can be used to identify the relevant parts of INLA predictor vector output, via bru_index().

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, weights, and scale) if not found in response_data or data. Defaults to the calling environment.

include, exclude, include_latent
: **[Deprecated]**, use used instead.

...
: For bru_obs_list.bru_obs, one or more bru_obs objects

.tag
: Optional name to assign to a single bru_obs object. Reserved for internal use.

object
: A list of bru_obs and/or bru_obs_list objects

x
: bru_obs_list object from which to extract element(s)

i
: indices specifying elements to extract

Details

The E, Ntrials, weights, and scale arguments are evaluated in the data context, with values from response_data taking precedence over data.

Value

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

Methods (by generic)

- bru_obs_list(bru_obs): Combine one or more lists of bru_obs observation model objects into a bru_obs_list object
- c(bru_obs): Combine several bru_obs objects into a bru_obs_list object

Functions

- like(): **[Deprecated]** Legacy like() method for inlabru prior to version 2.12.0. Use bru_obs() instead.
- bru_obs_list(): Combine bru_obs observation model object into a bru_obs_list object
- bru_obs_list(list): Combine one or more lists of bru_obs observation model objects into a bru_obs_list object
- bru_obs_list(bru_obs_list): Combine one or more bru_obs_list objects into a bru_obs_list object
- c(bru_obs_list): Combine several bru_obs_list objects into a bru_obs_list object
- like_list(): **[Deprecated]** Backwards compatibility for versions <= 2.12.0. For later versions, use as_bru_obs_list(), bru_obs_list(), or c().
- bru_like_list(): **[Deprecated]** Backwards compatibility for versions <= 2.12.0.9017. For later versions, use as_bru_obs_list(), bru_obs_list() or c().

Author(s)

Fabian E. Bachl <bachlfab@gmail.com>

Finn Lindgren <finn.lindgren@gmail.com>

See Also

bru_response_size(), bru_used(), bru_comp(), bru_comp_eval()

summary.bru_obs()

Examples

```
if (bru_safe_inla() &&
  require(ggplot2, quietly = TRUE) &&
  require(patchwork, quietly = TRUE)) {
  # The 'bru_obs()' (previously 'like()') function's main purpose is to set
  # up observation models, both for single- and multi-likelihood models.
  # 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 bru_obs
  # function

  lik1 <- bru_obs(
    "gaussian",
    formula = y ~ x + Intercept,
    data = df1,
    tag = "norm"
  )
  lik2 <- bru_obs(
    "poisson",
    formula = y ~ x + z + Intercept,
    data = df2,
    tag = "pois"
  )
```

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

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

  bru_index(jfit, "norm")
  bru_index(jfit, "pois")

  # 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")

  (p1 / p2 / pj)
}
```

`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()

bru_options_set_local(..., .reset = FALSE, .envir = parent.frame())
```

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).
.envir	The environment in which to set the options. Default: parent.frame()

Value

bru_options() returns a 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)).

Functions

- as.bru_options(): Coerces inputs to a bru_options object.
- bru_options_default(): Returns the default options.
- bru_options_check(): Checks for valid contents of a bru_options object, and produces warnings for invalid options.
- bru_options_get(): Used to access global package options.

- `bru_options_set()`: Used to set global package options.
- `bru_options_reset()`: Clears the global option overrides.
- `bru_options_set_local()`: Sets local option overrides, that are automatically reset using `withr::defer()`.

Valid options

For `bru_options` and `bru_options_set`, recognised options are:

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

bru_verbose_store numeric or logical, as for `bru_verbose`, but controls what messages are stored in the global log object. Default: `Inf`, to store all messages.

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

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

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 `fmesher::fm_int()` 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 `<not set>`.

bru_method List of arguments controlling the iterative inlabru method:

- **taylor** 'pandemic' (default, from version 2.1.15).
- **search** Either 'all' (default), to use all available line search methods, or one or more of
  - **'finite'** (reduce step size until predictor is finite)
  - **'contract'** (decrease step size until trust hypersphere reached)
  - **'expand'** (increase step size until no improvement)
  - **'optimise'** (fast approximate error norm minimisation)

  To disable line search, set to an empty vector. Line search is not available for `taylor="legacy"`.
- **factor** Numeric, > 1 determining the line search step scaling multiplier. Default $(1+\sqrt{5})/2$.
- **rel_tol** Stop the iterations when the largest change in linearisation point (the conditional latent state mode) in relation to the estimated posterior standard deviation is less than `rel_tol`. Default 0.1 (ten percent).
- **max_step** The largest allowed line search step factor. Factor 1 is the full INLA step. Default is 2.
- **line_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 either a single term, with $y = n$, and predictor mean(eta), or a blockwise version of this. Default: FALSE (was TRUE prior to version 2.13.0.9031.)

bru_debug logical; when TRUE, activate temporary debug features for package development. Default: FALSE

bru_compat_pre_2_14_enable logical; when TRUE, enable compatibility features for inlabru versions prior to 2.14. Set to FALSE to test external package compatibility updates. Default: TRUE before version 2.14, and will be set to FALSE by default in a later version.

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.

The following inla() options have inlabru specific defaults:

E Default 1.

Ntrials Default 1L.

control.compute Default list(config = TRUE, control.gcpo = list()).

control.inla Default list(int.strategy = "auto").

control.fixed Default list(expand.factor.strategy = "inla").

See Also

bru_options(), bru_options_default(), bru_options_get()

Examples

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

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

## End(Not run)
bru_options_get("bru_verbose")
## Not run:
if (interactive()) {
  bru_options_set(
```

```
    bru_verbose = TRUE,
    verbose = TRUE
  )
}

## End(Not run)
my_fun <- function(val) {
  bru_options_set_local(bru_verbose = val)
  bru_options_get("bru_verbose")
}
# Inside the function, the bru_verbose option is changed.
# Outside the function, the bru_verbose option is unchanged.
print(my_fun(TRUE))
print(bru_options_get("bru_verbose"))
print(my_fun(FALSE))
print(bru_options_get("bru_verbose"))
```

`bru_response_size` *Response size queries*

Description

Extract the number of response values from `bru` and related objects.

Usage

```
bru_response_size(object)

## Default S3 method:
bru_response_size(object)

## S3 method for class 'list'
bru_response_size(object)

## S3 method for class 'inla.surv'
bru_response_size(object)

## S3 method for class 'bru_obs'
bru_response_size(object)

## S3 method for class 'bru_obs_list'
bru_response_size(object)

## S3 method for class 'bru_info'
bru_response_size(object)

## S3 method for class 'bru'
bru_response_size(object)
```

Arguments

object	An object from which to extract response size(s).

Value

An integer vector.

Methods (by class)

- bru_response_size(default): Extract the number of observations from an object supporting NROW().
- bru_response_size(list): Extract the number of observations from an object supporting NROW(object[[1]]).
- bru_response_size(inla.surv): Extract the number of observations from an inla.surv object.
- bru_response_size(bru_obs): Extract the number of observations from a bru_obs object.
- bru_response_size(bru_obs_list): Extract the number of observations from a bru_obs_list object, as a vector with one value per observation model.
- bru_response_size(bru_info): Extract the number of observations from a bru_info object, as a vector with one value per observation model.
- bru_response_size(bru): Extract the number of observations from a bru object, as a vector with one value per observation model.

See Also

like()

Examples

```
bru_response_size(
  bru_obs(y ~ 1, data = data.frame(y = rnorm(10)), family = "gaussian")
)
```

---

bru_set_missing *Set missing values in observation models*

Description

Set all or parts of the observation model response data to NA, for example for use in cross validation (with bru_rerun()) or prior sampling (with bru_rerun() and generate(), but see "Prior sampling caveats" below).

Usage

```
bru_set_missing(object, keep = FALSE, ...)

bru_set_missing(x, ...) <- value

## S3 method for class 'bru'
bru_set_missing(object, keep = FALSE, ...)

## S3 method for class 'bru_info'
bru_set_missing(object, keep = FALSE, ...)

## S3 method for class 'bru_obs_list'
bru_set_missing(object, keep = FALSE, ...)

## S3 method for class 'bru_obs'
bru_set_missing(object, keep = FALSE, ...)

## Default S3 method:
bru_set_missing(object, keep = FALSE, ...)

## S3 method for class 'data.frame'
bru_set_missing(object, keep = FALSE, ...)

## S3 method for class 'inla.surv'
bru_set_missing(object, keep = FALSE, ...)
```

Arguments

object	A bru, bru_obs or bru_obs_list object
keep	For bru_obs, a single logical or an integer vector; If TRUE, keep all the response data, if FALSE (default), set all of it to NA. An integer vector determines which elements to keep (for positive values) or to set as missing (negative values). For bru and bru_obs_list, a logical scalar or vector, or a list, see Details.
...	Additional arguments passed on to the bru_obs and individual data class methods. Currently unused.
x	Object on which to apply bru_set_missing()
value	Value to be passed as keep to bru_set_missing()

Details

For bru and bru_obs_list,

- keep must be either a single logical, which is expanded to a list,
- a logical vector, which is converted to a list,
- an unnamed list of the same length as the number of observation models, with elements compatible with the bru_obs method, or

- a named list with elements compatible with the `bru_obs` method, and only the named `bru_obs` models are acted upon, i.e. the elements not present in the list are treated as `keep = TRUE`.

E.g.: `keep = list(b = FALSE)` sets all observations in model `b` to missing, and does not change model `a`.

E.g.: `keep = list(a = 1:4, b = -(3:5))` keeps only observations `1:4` of model `a`, marking the rest as missing, and sets observations `3:5` of model `b` to missing.

Methods (by class)

- `bru_set_missing(default)`: From `> 2.13.0`, set missing values in any object supporting `base::is.na<-()` for positive and negative indices.
- `bru_set_missing(data.frame)`: From `> 2.13.0`, handles `data.frame`, tibbles, including `inla.mdata`.
- `bru_set_missing(inla.surv)`: From `> 2.13.0`, handles `inla.surv`.

Functions

- `bru_set_missing(x, ...) <- value`: Setter method for `bru_set_missing()`

Prior sampling caveats

Note that prior sampling requires special care for hyperparameters, as the prior modes are not typically useful; in the future, we plan to have a dedicated method that samples from the hyperparameters, and then uses

```
bru_rerun(
  bru_set_missing(...),
  options = list(
    control.mode = list(
      theta = theta_sample,
      fixed = TRUE)
    )
  )
```

for each sample.

Examples

```
obs <- c(
  A = bru_obs(y_A ~ ., data = data.frame(y_A = 1:6)),
  B = bru_obs(y_B ~ ., data = data.frame(y_B = 11:15))
)
bru_response_size(obs)
lapply(
  bru_set_missing(obs, keep = FALSE),
  function(x) {
    x[["response_data"]][[x["response"]]]
  }
```

```
)
lapply(
  bru_set_missing(obs, keep = list(B = FALSE)),
  function(x) {
    x[["response_data"]][[x[["response"]]]]
  }
)
lapply(
  bru_set_missing(obs, keep = list(1:4, -(3:5))),
  function(x) {
    x[["response_data"]][[x[["response"]]]]
  }
)


(obs <- INLA::inla.mdata(y = 1:4, X = matrix(1:8, 4, 2)))
bru_set_missing(obs, keep = c(1, 4))
bru_set_missing(obs) <- -(1:2)
obs


(obs <- INLA::inla.surv(time = 1:4, event = c(1, 0, 1, 0)))
bru_set_missing(obs, keep = c(1, 4))

(obs <- INLA::inla.surv(
  time = 1:4,
  event = c(1, 0, 1, 0),
  cure = matrix(1:8, 4, 2)
))
bru_set_missing(obs, keep = c(1, 4))

(obs <- INLA::inla.surv(
  time = 1:4,
  event = c(1, 0, 1, 0),
  subject = c(1, 1, 2, 1)
))
bru_set_missing(obs, keep = c(1, 4))
```

`bru_timings` *Extract timing information from fitted bru object*

Description

Extracts a data.frame or tibble with information about the `Time` (CPU), `System`, and `Elapsed` time for each step of a `bru()` run.

Usage

```
bru_timings(object, ...)
```

```
## S3 method for class 'bru'
bru_timings(object, ...)
```

Arguments

object	A fitted bru object
...	unused

`bru_timings_plot` *Plot inlabru iteration timings*

Description

Draws the time per iteration for preprocessing (including linearisation), `inla()` calls, and line search. Iteration `0` is the time used for defining the model structure.

Usage

```
bru_timings_plot(x)
```

Arguments

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

Details

Requires the "ggplot2" package to be installed.

Examples

```
## Not run:
fit <- bru(...)
bru_timings_plot(fit)

## End(Not run)
```

`bru_transformation` *Transformation tools*

Description

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

Usage

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

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

Arguments

`qfun`	A quantile function object, such as `qexp`
`x`	Values to be transformed
`...`	Distribution parameters passed on to the `qfun` and `pfun` functions
`tail.split.`	For x-values larger than `tail.split.`, upper quantile calculations are used internally, and for smaller values lower quantile calculations are used. This can avoid lack of accuracy in the distribution tails. If `NULL`, forward calculations split at 0, and inverse calculations use lower tails only, potentially losing accuracy in the upper tails.
`pfun`	A CDF function object, such as `pexp`

Value

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

Examples

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

deltaIC *Summarise DIC from* lgcp *estimates.*

Description

Calculates DIC differences and produces an ordered summary.

Usage

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

Arguments

...	Comma-separated objects inheriting from class inla and obtained from a run of INLA::inla(), bru() or lgcp()
criterion	character vector. If it includes 'DIC', computes DIC differences; 'WAIC' is also allowed, but note that plain WAIC values from inla are not well-defined for point process models. Default 'WAIC'

Value

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

Examples

```
if (bru_safe_inla()) {
  # Generate some data
  input.df <- data.frame(idx = 1:10) |>
    dplyr::mutate(
      x = cos(idx),
      y = rpois(length(x), 5 + 2 * x + rnorm(length(x), mean = 0, sd = 0.1))
    )

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

  # 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`	An `sf` or `SpatialPolygon` object describing the area for which to compute the cumulative variance measure.
`mesh`	The fmesher::fm_mesh_2d for which the prediction was performed (required for cumulative Vmeasure).

Value

Variance and correlations measures.

References

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

Examples

```
if (bru_safe_inla() &&
  require("ggplot2", quietly = TRUE) &&
  require("patchwork", quietly = TRUE) &&
  requireNamespace("sn", quietly = TRUE) &&
  bru_safe_terra(quietly = TRUE) &&
  require("sf", quietly = TRUE) &&
  require("RColorBrewer", quietly = TRUE) &&
  require("dplyr", quietly = TRUE)) {
  # Load Gorilla data
```

```
gorillas <- gorillas_sf
gorillas$gcov <- gorillas_sf_gcov()

# Use RColorBrewer

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

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

cmp <- geometry ~ 0 +
  vegetation(gorillas$gcov$vegetation, model = "factor_contrast") +
  spde(geometry, model = pcmatern) +
  Intercept(1)

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

# Predict SPDE and vegetation at a grid covering the domain of interest
pred_loc <- fmesher::fm_pixels(
  gorillas$mesh,
  mask = gorillas$boundary,
  dims = c(200, 200),
  format = "sf"
)
pred <- predict(
  fit,
  pred_loc,
  ~ list(
    joint = spde + vegetation,
    field = spde,
    veg = vegetation
  )
)

pred_collect <-
  rbind(
    pred$joint |> dplyr::mutate(component = "joint"),
    pred$field |> dplyr::mutate(component = "field"),
    pred$veg |> dplyr::mutate(component = "veg")
  ) |>
  dplyr::mutate(var = sd^2)
```

```
# Plot component mean

ggplot(pred_collect) +
  geom_tile(aes(geometry = geometry, fill = mean),
    stat = "sf_coordinates"
  ) +
  coord_equal() +
  facet_wrap(~component, nrow = 1) +
  theme(legend.position = "bottom")

# Plot component variance

ggplot(pred_collect) +
  geom_tile(aes(geometry = geometry, fill = var),
    stat = "sf_coordinates"
  ) +
  coord_equal() +
  facet_wrap(~component, nrow = 1) +
  theme(legend.position = "bottom")

# Calculate variance and correlation measure

vm <- devel.cvmeasure(pred$joint, pred$field, pred$veg)
# Compute nominal relative variance contributions; note that these can be
# greater than 100%!
vm <- dplyr::mutate(
  vm,
  var1_rel = if_else(var1 <= 0, NA, var1 / var.joint),
  var2_rel = if_else(var2 <= 0, NA, var2 / var.joint)
)
lprange <- range(vm$var.joint, vm$var1, vm$var2)
vm <- tidyr::pivot_longer(vm,
  cols = c(var.joint, var1, var2, cov, cor, var1_rel, var2_rel),
  names_to = "component",
  values_to = "value"
)

# Variance contribution of the components

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

vm_ <- dplyr::filter(vm, component %in% c("var.joint", "var1", "var2"))
ggplot(vm_) +
  geom_tile(aes(geometry = geometry, fill = value),
    stat = "sf_coordinates"
  ) +
  csc +
  coord_equal() +
  facet_wrap(~component, nrow = 1) +
  theme(legend.position = "bottom")
```

```
# Relative variance contribution of the components
# When bo

vm_ <- dplyr::filter(vm, component %in% c("var1_rel", "var2_rel"))
ggplot(vm_) +
  geom_tile(aes(geometry = geometry, fill = value),
    stat = "sf_coordinates"
  ) +
  scale_fill_gradientn(
    colours = brewer.pal(9, "YlOrRd"),
    trans = "log"
  ) +
  coord_equal() +
  facet_wrap(~component, nrow = 1)

# Where both relative contributions are larger than 1, the posterior
# correlations are strongly negative.

# Covariance and correlation of field and vegetation

vm_cov <- dplyr::filter(vm, component %in% "cov")
vm_cor <- dplyr::filter(vm, component %in% "cor")
(ggplot(vm_cov) +
  geom_tile(aes(geometry = geometry, fill = value),
    stat = "sf_coordinates"
  ) +
  scale_fill_gradientn(
    colours = rev(brewer.pal(9, "RdBu")),
    limits = c(-1, 1) * max(abs(vm_cov$value), na.rm = TRUE)
  ) +
  coord_equal() +
  theme(legend.position = "bottom") +
  ggtitle("Covariances") |
  ggplot(vm_cor) +
    geom_tile(aes(geometry = geometry, fill = value),
      stat = "sf_coordinates"
    ) +
    scale_fill_gradientn(
      colours = rev(brewer.pal(9, "RdBu")),
      limits = c(-1, 1) * max(abs(vm_cor$value), na.rm = TRUE)
    ) +
    coord_equal() +
    theme(legend.position = "bottom") +
    ggtitle("Correlations")
)

# Variance and correlation integrated over space

vrt <- fmesher::fm_vertices(gorillas$mesh, format = "sf")
pred_vrt <- predict(
  fit,
  vrt,
```

```
    ~ list(
      joint = spde + vegetation,
      field = spde,
      veg = vegetation
    )
  )

  vm.int <- devel.cvmeasure(
    pred_vrt$joint,
    pred_vrt$field,
    pred_vrt$veg,
    samplers = fmesher::fm_int(gorillas$mesh, gorillas$boundary),
    mesh = gorillas$mesh
  )
  vm.int
}
```

`eval_spatial` *Evaluate spatial covariates*

Description

Evaluate spatial covariates

Usage

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

## S3 method for class 'SpatialPolygonsDataFrame'
eval_spatial(data, where, layer = NULL, selector = NULL)

## S3 method for class 'SpatialPixelsDataFrame'
eval_spatial(data, where, layer = NULL, selector = NULL)

## S3 method for class 'SpatialGridDataFrame'
eval_spatial(data, where, layer = NULL, selector = NULL)

## S3 method for class 'sf'
eval_spatial(data, where, layer = NULL, selector = NULL)

## S3 method for class 'SpatRaster'
eval_spatial(data, where, layer = NULL, selector = NULL)

## S3 method for class 'stars'
eval_spatial(data, where, layer = NULL, selector = NULL)
```

Arguments

data	Spatial 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 item.
selector	The name of a variable in where specifying the layer information.

Methods (by class)

- eval_spatial(SpatialPolygonsDataFrame): Compatibility wrapper for eval_spatial.sf
- eval_spatial(sf): Supports point-in-polygon information lookup. Other combinations are untested.

---

format.bru_input *Summarise component inputs*

Description

Summarise component inputs

Usage

```
## S3 method for class 'bru_input'
format(x, verbose = TRUE, ..., label.override = NULL, type = NULL)

## S3 method for class 'bru_input'
summary(object, verbose = TRUE, ..., label.override = NULL)

## S3 method for class 'bru_input'
print(x, verbose = TRUE, ..., label.override = NULL)
```

Arguments

x	Object to be printed
verbose	logical; If TRUE, includes more details of the component definitions. When FALSE, only show basic component definition information. Default TRUE.
...	Passed on to other summary methods.
label.override	character; If not NULL, use this label instead of the object's label.
type	character; if non-NULL, added to the output'; label = type(input).
object	Object to be summarised.

Author(s)

Fabian E. Bachl <bachlfab@gmail.com>

Finn Lindgren <finn.lindgren@gmail.com>

See Also

bru_input(), bru_comp()

---

format.bru_mapper *mapper object summaries*

Description

mapper object summaries

Usage

```
## S3 method for class 'bru_mapper'
format(x, ..., prefix = "", initial = prefix, depth = 1)

## S3 method for class 'bm_list'
format(
  x,
  ...,
  prefix = "",
  initial = prefix,
  depth = 1,
  collapse = ", ",
  labels = TRUE
)

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

## S3 method for class 'bm_multi'
format(x, ..., prefix = "", initial = prefix, depth = 1)

## S3 method for class 'bm_pipe'
format(x, ..., prefix = "", initial = prefix, depth = 1)

## S3 method for class 'bm_collect'
format(x, ..., prefix = "", initial = prefix, depth = 1)

## S3 method for class 'bm_sum'
format(x, ..., prefix = "", initial = prefix, depth = 1)

## S3 method for class 'bm_repeat'
format(x, ..., prefix = "", initial = prefix, depth = 1)

## S3 method for class 'bm_reparam'
format(x, ..., prefix = "", initial = prefix, depth = 1)
```

```
## S3 method for class 'summary_bru_mapper'
print(x, ..., sep = "\n")

## S3 method for class 'bru_mapper'
print(x, ..., sep = "\n", prefix = "", initial = prefix, depth = 1)

## S3 method for class 'bm_list'
print(
  x,
  ...,
  sep = "\n",
  prefix = "",
  initial = prefix,
  depth = 1,
  labels = TRUE,
  collapse = ", "
)
```

Arguments

x	Object to format/print
...	Unused arguments
prefix	character prefix for each line. Default "".
initial	character prefix for the first line. Default initial=prefix.
depth	The recursion depth for multi/collection/pipe mappers. Default 1, to only show the collection, and not the contents of the sub-mappers.
collapse	character or NULL, as in base::paste().
labels	logical; if TRUE, include mapper names or numerical indices. Default TRUE
object	Object to summarise
sep	character; separator for printing the summary.

Value

- format character.

Examples

```
mapper <-
  bm_pipe(
    list(
      bm_multi(list(
        A = bm_index(2),
        B = bm_index(3)
      )),
      bm_index(2)
    )
  )
```

```
summary(mapper, depth = 2)
mapper <-
  bm_repeat(
    bm_multi(
      list(
        A = bm_index(2),
        B = bm_index(3)
      )
    ),
    3
  )
summary(mapper)
summary(mapper, depth = 0)
mapper <-
  bm_reparam(
    bm_multi(
      list(
        A = bm_index(2),
        B = bm_index(3)
      )
    ),
    matrix(1:36, nrow = 6)
  )
summary(mapper)
summary(mapper, depth = 0)
```

`generate` *Generate samples from fitted bru models*

Description

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

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

Usage

```
generate(object, ...)

## S3 method for class 'bru'
generate(
  object,
  newdata = NULL,
  formula = NULL,
  n.samples = 100,
  seed = 0L,
```

```
  num.threads = NULL,
  used = NULL,
  ...,
  include = deprecated(),
  exclude = deprecated()
)
```

Arguments

object	A bru object obtained by calling bru().
...	additional, unused arguments.
newdata	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 "1:1:1"
used	Either NULL or a bru_used() object. Default, NULL, uses auto-detection of used variables in the formula.
include, exclude	**[Deprecated]** If auto-detection of used variables fails, use used instead.

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 bru_comp_eval()).

For "iid" models with mapper = bm_index(n), rnorm() is used to generate new realisations for indices greater than n, if accessed via <name>_eval(...).

Value

The form of the value returned by generate() depends on the data class and prediction formula. Normally, a data.frame is returned, or a list of data.frames (if the prediction formula generates a list)

List of generated samples

See Also

predict.bru

Examples

```
if (
  bru_safe_inla() &&
    requireNamespace("sn", quietly = TRUE)
) {
  # 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

```
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: gg.RasterLayer(), gg.SpatRaster(), gg.Spatial, gg.data.frame(), gg.fm_mesh_1d(), gg.fm_mesh_2d(), gg.matrix(), gg.sf()

Examples

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

  gorillas <- inlabru::gorillas_sf

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

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

gg.data.frame *Geom for 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.bru_prediction also understands the format of INLA-style posterior summaries, e.g. fit$summary.fixed for an inla object fit

Requires the ggplot2 package.

Usage

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

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

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

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

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

Arguments

...	Arguments passed on to geom_line.
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(). If "fill" is present, it is passed to geom_ribbon().
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.
x	a prediction object.
y	Ignored argument but required for S3 compatibility.

Value

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

Functions

- `gg(data.frame)`: This geom constructor will simply call `gg.bru_prediction()` for the data provided.
- `plot(bru_prediction)`: Generates a base ggplot2 using `ggplot()` and adds a geom for input `x` using `gg.bru_prediction()`. Returns a ggplot object.
- `plot(prediction)`: Identical to `gg.bru_prediction()`.

See Also

Other geomes: `gg()`, `gg.RasterLayer()`, `gg.SpatRaster()`, `gg.Spatial`, `gg.fm_mesh_1d()`, `gg.fm_mesh_2d()`, `gg.matrix()`, `gg.sf()`

Examples

```
if (bru_safe_inla() &&
  requireNamespace("sn", quietly = TRUE) &&
  require("ggplot2", quietly = TRUE) &&
  require("patchwork", 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, 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,
    newdata = 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)
  (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 of using the \code{ribbon} parameter

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

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

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

gg.fm_mesh_1d *Geom for fm_mesh_1d objects*

Description

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

Usage

```
## S3 method for class 'fm_mesh_1d'
gg(
  data,
  mapping = ggplot2::aes(.data[["x"]], .data[["y"]]),
  y = 0,
  shape = 4,
  ...
)
```

Arguments

data	An fmesher::fm_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: gg(), gg.RasterLayer(), gg.SpatRaster(), gg.Spatial, gg.data.frame(), gg.fm_mesh_2d(), gg.matrix(), gg.sf()

Examples

```
if (require("fmesher", quietly = TRUE) &&
  require("ggplot2", quietly = TRUE)) {
  # Create a 1D mesh

  mesh <- fmesher::fm_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.fm_mesh_2d` *Geom for fm_mesh_2d objects*

Description

This function extracts the graph of an fmesher::fm_mesh_2d 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.

Also see the `fmesher::geom_fm()` method.

Usage

```
## S3 method for class 'fm_mesh_2d'
gg(
  data,
  color = NULL,
  alpha = NULL,
  edge.color = "grey",
  edge.linewidth = 0.25,
  interior = TRUE,
  int.color = "blue",
  int.linewidth = 0.5,
  exterior = TRUE,
  ext.color = "black",
  ext.linewidth = 1,
  crs = NULL,
  mask = NULL,
  nx = 500,
  ny = 500,
  ...
)
```

Arguments

data	An fm_mesh_2d object.
color	A vector of scalar values to fill the mesh with colors. The length of the vector mus 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 regular mesh edges.
edge.linewidth	Line width for the regular mesh edges. Default 0.25
interior	If TRUE, plot the interior boundaries of the mesh.
int.color	Color used to plot the interior constraint edges.
int.linewidth	Line width for the interior constraint edges. Default 0.5
exterior	If TRUE, plot the exterior boundaries of the mesh.
ext.color	Color used to plot the exterior boundary edges.
ext.linewidth	Line width for the exterior boundary edges. Default 1
crs	A CRS object supported by fmesher::fm_transform() defining the coordinate system to project the mesh to before plotting.
mask	A SpatialPolygon or sf polygon 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: gg(), gg.RasterLayer(), gg.SpatRaster(), gg.Spatial, gg.data.frame(), gg.fm_mesh_1d(), gg.matrix(), gg.sf()

Examples

```
if (require(fmesher, quietly = TRUE) &&
  require(ggplot2, quietly = TRUE)) {
  # Load Gorilla data
  gorillas <- inlabru::gorillas_sf

  # Plot mesh using default edge colors

  ggplot() +
    gg(gorillas$mesh)

  # Don't show interior and exterior boundaries

  ggplot() +
```

```
  gg(gorillas$mesh, interior = FALSE, exterior = FALSE)

# Change the edge colors

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

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

xcoord <- gorillas$mesh$loc[, 1]
ggplot() +
  gg(gorillas$mesh, color = (xcoord - 580), mask = gorillas$boundary) +
  gg(gorillas$boundary, alpha = 0)
}
```

`gg.matrix` *Geom for matrix*

Description

Creates a tile geom for plotting a matrix

Usage

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

Arguments

`data`	A `matrix` object.
`mapping`	a set of aesthetic mappings created by `aes`. These are passed on to `geom_tile`.
`...`	Arguments passed on to `geom_tile`.

Details

Requires the `ggplot2` package.

Value

A `geom_tile` with reversed y scale.

See Also

Other geomes: gg(), gg.RasterLayer(), gg.SpatRaster(), gg.Spatial, gg.data.frame(), gg.fm_mesh_1d(), gg.fm_mesh_2d(), gg.sf()

Examples

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

---

gg.RasterLayer *Geom for RasterLayer objects*

Description

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

Usage

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

Arguments

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

Details

This function requires the raster and ggplot2 packages.

Value

An object returned by geom_tile

See Also

Other geomes: gg(), gg.SpatRaster(), gg.Spatial, gg.data.frame(), gg.fm_mesh_1d(), gg.fm_mesh_2d(), gg.matrix(), gg.sf()

Examples

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

  # Convert elevation covariate to RasterLayer

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

  # Plot the elevation

  ggplot() +
    gg(elev)
}

## End(Not run)
```

`gg.sf` *Geom helper for sf objects*

Description

This function uses `geom_sf()`, unless overridden by the geom argument. Requires the `ggplot2` package.

Usage

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

Arguments

`data`	An `sf` object.
`mapping`	Default mapping is `ggplot2::aes(geometry = ...)`, where the geometry name is obtained from `attr(data, "sf_column")`. This is merged with the user supplied mapping.
`...`	Arguments passed on to `geom_sf` or `geom_tile`.
`geom`	Either "sf" (default) or "tile". For "tile", uses `geom_tile(..., stat = "sf_coordinates")`, intended for converting point data to grid tiles with the `fill` aesthetic, which is by default set to the first data column.

Value

A ggplot return value

See Also

Other geomes: gg(), gg.RasterLayer(), gg.SpatRaster(), gg.Spatial, gg.data.frame(), gg.fm_mesh_1d(), gg.fm_mesh_2d(), gg.matrix()

Examples

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

  gorillas <- inlabru::gorillas_sf
  gorillas$gcov <- gorillas_sf_gcov()

  # Plot Gorilla elevation covariate provided as terra::rast.

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

  # Add Gorilla survey boundary and nest sightings

  ggplot() +
    gg(gorillas$gcov$elevation) +
    gg(gorillas$boundary, alpha = 0) +
    gg(gorillas$nests)

  # Load pantropical dolphin data

  mexdolphin <- inlabru::mexdolphin_sf

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

  ggplot() +
    gg(mexdolphin$ppoly, alpha = 0.5) + # survey boundary
    gg(mexdolphin$samplers) + # ship transects
    gg(mexdolphin$points) # dolphin sightings

  # Change color

  ggplot() +
    gg(mexdolphin$ppoly, color = "green", alpha = 0.5) + # survey boundary
    gg(mexdolphin$samplers, color = "red") + # ship transects
    gg(mexdolphin$points, color = "blue") # dolphin sightings


  # 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.Spatial *Geoms for sp Spatial objects*

Description

Methods for plotting sp spatial objects with ggplot2.

Usage

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

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

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

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

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

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

Arguments

data	A Spatial* object.

mapping	Aesthetic mappings created by aes used to update the default mapping. Uness specified otherwise below, the default mapping is
	`ggplot2::aes(`
	`  x = .data[[sp::coordnames(data)[1]]],`
	`  y = .data[[sp::coordnames(data)[2]]]`
	`)`

crs	A sp::CRS object defining the coordinate system to project the data to before plotting.

...	Arguments passed on to geom_*.
mask	A sp::SpatialPolygons object defining the region that is plotted.

Value

A geom_point, geom_segment, geom_sf, geom_tile, or a list of ggplot geomes

Functions

- gg(SpatialPoints): Geom for SpatialPoints objects. This function coerces the SpatialPoints into a data.frame and uses geom_point to plot the points. Requires the ggplot2 package.
- gg(SpatialLines): Geom for SpatialLines objects.

  Extracts start and end points of the lines and calls geom_segment to plot lines between them.

  mapping: Aesthetic mappings created by ggplot2::aes or ggplot2::aes_ used to update the default mapping. The default mapping is

  ```
  ggplot2::aes(
    x = .data[[sp::coordnames(data)[1]]],
    y = .data[[sp::coordnames(data)[2]]],
    xend = .data[[paste0("end.", sp::coordnames(data)[1])]],
    yend = .data[[paste0("end.", sp::coordnames(data)[2])]])
  ```

- gg(SpatialPolygons): Geom for SpatialPolygons objects. Uses the ggplot2::fortify() function to turn the SpatialPolygons objects into a data.frame. Then calls geom_polygon to plot the polygons.

  Unless specified by the user, the argument alpha = 0.2 (alpha level for polygon filling) is added.

  Up to version 2.10.0, the ggpolypath package was used to ensure proper plotting for polygons, since the ggplot2::geom_polygon function doesn't always handle geometries with holes properly. After 2.10.0, the object is converted to sf format and passed on to gg.sf() instead, as ggplot2 version 3.4.4 deprecated the internally used ggplot2::fortify() method for SpatialPolygons/DataFrame objects.
- gg(SpatialGridDataFrame): Geom for SpatialGridDataFrame objects

  Coerces input SpatialGridDataFrame to SpatialPixelsDataFrame and calls gg.SpatialPixelsDataFrame() to plot it.
- gg(SpatialPixelsDataFrame): Geom for SpatialPixelsDataFrame objects.

  Coerces SpatialPixelsDataFrame input to data.frame and uses geom_tile to plot it.

  mapping: Aesthetic mappings created by aes used to update the default mapping. The default mapping is

  ```
  ggplot2::aes(
    x = .data[[sp::coordnames(data)[1]]],
    y = .data[[sp::coordnames(data)[2]]],
    fill = .data[[names(data)[[1]]]]
  )
  ```

- gg(SpatialPixels): Geom for SpatialPixels objects

  Converts the input to SpatialPoints and calls [gg.SpatialPoints()‘ to plot it.

See Also

Other geomes: gg(), gg.RasterLayer(), gg.SpatRaster(), gg.data.frame(), gg.fm_mesh_1d(), gg.fm_mesh_2d(), gg.matrix(), gg.sf()

Examples

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

  gorillas <- inlabru::gorillas_sf

  gcov <- gorillas_sf_gcov()
  elev <- terra::as.data.frame(gcov$elevation, xy = TRUE)
  elev <- sf::as_Spatial(sf::st_as_sf(elev, coords = c("x", "y")))

  # Turn elevation covariate into SpatialGridDataFrame
  elev <- sp::SpatialPixelsDataFrame(elev, data = as.data.frame(elev))

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

  ggplot() +
    gg(elev)

  # Add Gorilla survey boundary and nest sightings

  ggplot() +
    gg(elev) +
    gg(gorillas$boundary, alpha = 0.0, col = "red") +
    gg(gorillas$nests)

  # Load pantropical dolphin data

  mexdolphin <- inlabru::mexdolphin_sp()

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


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

  gcov <- gorillas_sf_gcov()
  elev <- terra::as.data.frame(gcov$elevation, xy = TRUE)
  pxl <- sf::as_Spatial(sf::st_as_sf(elev, coords = c("x", "y")))

  # Turn elevation covariate into SpatialPixels
  pxl <- sp::SpatialPixels(pxl)

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

`gg.SpatRaster` *Geom wrapper for SpatRaster objects*

Description

Convenience wrapper function for `tidyterra::geom_spatraster()`. Requires the `ggplot2` and `tidyterra` packages.

Usage

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

Arguments

`data`	A SpatRaster object.
`...`	Arguments passed on to `geom_spatraster`.

Value

The output from 'geom_spatraster.

See Also

Other geomes: gg(), gg.RasterLayer(), gg.Spatial, gg.data.frame(), gg.fm_mesh_1d(), gg.fm_mesh_2d(), gg.matrix(), gg.sf()

Examples

```
if (require("ggplot2", quietly = TRUE) &&
  bru_safe_terra(quietly = TRUE) &&
  require("tidyterra", quietly = TRUE)) {
  # Load Gorilla covariates

  gcov <- gorillas_sf_gcov()

  # Plot the pixel centers
  ggplot() +
    gg(gcov$elevation)
}
```

---

glance.bru *Glance at a bru model fit*

Description

Glance at a bru model fit

Usage

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

Arguments

x	A fitted bru object.
...	Unused.

Value

A one-row tibble of model-level summaries. elapsed is wall-clock seconds summed across the phases in fit$bru_timings, not CPU time. nobs is the total row count summed across all likelihoods, returned as NA for models that involve family = "cp", as "observation count" is not meaningful for such models.

glplot *Render objects using RGL*

Description

`glplot()` is a generic function for renders various kinds of spatial objects, i.e. `Spatial*` data and `fm_mesh_2d` objects. The function invokes particular methods which depend on the class of the first argument.

Usage

```
glplot(object, ...)

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

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

## S3 method for class 'fm_mesh_2d'
glplot(object, add = TRUE, col = NULL, ...)

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

Arguments

`object`	an object used to select a method.
`...`	Parameters passed on to `plot_rgl.fm_mesh_2d()`
`add`	If TRUE, add the points to an existing plot. If FALSE, create new plot.
`color`	vector of R color characters. See material3d() for details.
`col`	Color specification. A single named color, a vector of scalar values, or a matrix of RGB values.
`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

Methods (by class)

- glplot(SpatialPoints): This function will calculate the cartesian coordinates of the points provided and use points3d() in order to render them.
- glplot(SpatialLines): This function will calculate a cartesian representation of the lines provided and use lines3d() in order to render them.
- glplot(fm_mesh_2d): This function transforms the mesh to 3D cartesian coordinates and uses fmesher::plot_rgl() to plot the result.

Functions

- globe(): Visualize a globe using RGL
  Creates a textured sphere and lon/lat coordinate annotations. This function requires the rgl and sphereplot packages.

Examples

```
if (interactive() &&
  require("rgl", quietly = TRUE) &&
  require("sphereplot", quietly = TRUE) &&
  bru_safe_sp() &&
  require("sp")) {
  # Show the globe:
  globe()

  # Load pantropoical dolphin data
  mexdolphin <- inlabru::mexdolphin_sp()

  # Add mesh, ship transects and dolphin sightings stored
  # as fm_mesh_2d, SpatialLines and SpatialPoints objects, respectively

  glplot(mexdolphin$mesh, alpha = 0.2)
  glplot(mexdolphin$samplers, lwd = 5)
  glplot(mexdolphin$points, size = 10)
}
```

---

gorillas_sf *Gorilla nesting sites in sf format*

Description

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

Usage

```
gorillas_sf
data(gorillas_sf, package = "inlabru")

gorillas_sf_gcov()

gorillas_sp()
```

Format

The data are a list that contains these elements:

`nests`: An `sf` object containing the locations of the gorilla nests.

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

`mesh`: An `fm_mesh_2d` object containing a mesh that can be used with function `lgcp` to fit a LGCP to the nest data.

`gcov_file`: The in-package filename of a `terra::SpatRaster` object, with one layer for each of these spatial covariates:

- `aspect` Compass direction of the terrain slope. Categorical, with levels N, NE, E, SE, S, SW, W and NW, which are coded as integers 1 to 8.
- `elevation` Digital elevation of terrain, in metres.
- `heat` Heat Load Index at each point on the surface (Beer's aspect), discretised. Categorical with values Warmest (Beer's aspect between 0 and 0.999), Moderate (Beer's aspect between 1 and 1.999), Coolest (Beer's aspect equals 2). These are coded as integers 1, 2 and 3, in that order.
- `slopangle` Terrain slope, in degrees.
- `slopetype` 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.

Loading of the covariates can be done with `gorillas_sf_gcov()` or

```
gorillas_sf$gcov <- terra::rast(
  system.file(gorillas_sf$gcov_file, package = "inlabru")
)
```

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

- `counts` An `sf` object with elements `count`, `exposure`, and `geometry`, holding the point geometry for the centre of each plot, the count in each plot and the area of each plot.
- `plots` An `sf` object with `MULTIPOLYGON` objects defining the individual plot boundaries and an all-ones `weight` column.
- `nests` An `sf` giving the locations of each detected nests, `group` ("minor" or "major"), `season` ("dry" or "rainy"), and `date` (in `Date` format).

Functions

- `gorillas_sf_gcov()`: Access the `gorillas_sf` covariates data as a `terra::rast()` object.
- `gorillas_sp()`: Access the `gorillas_sf` data in `sp` format. The covariate data is added as `gcov`, a list of `sp::SpatialPixelsDataFrame` objects. Requires the `sp`, `sf`, and `terra` packages to be installed.

Source

Library `spatstat.data`.

References

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

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

Examples

```
if (interactive() &&
  require(ggplot2, quietly = TRUE) &&
  bru_safe_terra(quietly = TRUE) &&
  requireNamespace("tidyterra", quietly = TRUE)) {
  # plot all the nests, mesh and boundary
  ggplot() +
    gg(gorillas_sf$mesh) +
    geom_sf(
      data = gorillas_sf$boundary,
      alpha = 0.1, fill = "blue"
    ) +
    geom_sf(data = gorillas_sf$nests)

  # Plot the elevation covariate
  gorillas_sf$gcov <- gorillas_sf_gcov()
  ggplot() +
    tidyterra::geom_spatraster(data = gorillas_sf$gcov$elevation)

  # Plot the plot sample
  ggplot() +
    geom_sf(data = gorillas_sf$plotsample$plots) +
    geom_sf(data = gorillas_sf$plotsample$nests)
}
if (interactive() &&
  bru_safe_terra(quietly = TRUE)) {
  gorillas_sf$gcov <- gorillas_sf_gcov()
}
```

ibm_eval *Evaluate a mapping*

Description

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.

Specific implementations for `bm_aggregate`, `bm_collect`, `bm_const`, `bm_factor`, `bm_fm_mesh_1d`, `bm_fm_mesh_2d`, `bm_fmesher`, `bm_harmonics`, `bm_index`, `bm_inla_mesh_1d`, `bm_inla_mesh_2d`, `bm_linear`, `bm_logitaverage`, `bm_logsumexp`, `bm_marginal`, `bm_matrix`, `bm_multi`, `bm_pipe`, `bm_reparam`, `bm_repeat`, `bm_scale`, `bm_shift`, `bm_sum`, `bm_taylor`, `default`.

Usage

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

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

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

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

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

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

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

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

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

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

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

```
## S3 method for class 'bm_multi'
ibm_eval(
  mapper,
  input,
  state = NULL,
  inla_f = FALSE,
  ...,
  jacobian = NULL,
  pre_A = deprecated()
)

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

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

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

## S3 method for class 'bm_sum'
ibm_eval(mapper, input, state, multi = FALSE, ..., sub_lin = NULL)
```

Arguments

mapper	A mapper S3 object, inheriting from bru_mapper.
input	Data input for the mapper.
state	A vector of latent state values for the mapping, of length ibm_n(mapper, inla_f = FALSE)
...	Arguments passed on to other methods
jacobian	For ibm_eval() methods, an optional pre-computed Jacobian, typically supplied by internal methods that already have the Jacobian.
log	logical; control log output. Default TRUE, see the ibm_eval() details for logsumexp mappers.
logit	logical; control logit output. Default TRUE, see the ibm_eval() details for logitaverage mappers.
reverse	logical; control bm_marginal evaluation. Default FALSE. When TRUE, reverses the direction of the mapping, see details for marginal mappers.

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 bm_collect.
pre_A	**[Deprecated]** in favour of jacobian.
multi	logical; If TRUE (or positive), recurse one level into sub-mappers
sub_lin	Internal, optional pre-computed sub-mapper information

Methods (by class)

- 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_eval(bm_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.
- ibm_eval(bm_const): Returns the input values, with NA replaced by 0.
- ibm_eval(bm_logsumexp): When log is TRUE (default), ibm_eval() for logsumexp returns the log-sum-weight-exp value. If FALSE, the sum-weight-exp value is returned.
- ibm_eval(bm_logitaverage): When logit is TRUE (default), ibm_eval() for logitaverage returns the logit-sum-weight-inverse-logit value. If FALSE, the sum-weights=invere-logit value is returned.
- ibm_eval(bm_marginal): When xor(mapper[["inverse"]], reverse) is FALSE, ibm_eval() for marginal returns qfun(pnorm(x), param), evaluated in a numerically stable way. Otherwise, evaluates the inverse qnorm(pfun(x, param)) instead.

See Also

Other mapper methods: bru_mapper_generics, ibm_eval2(), ibm_inla_subset(), ibm_invalid_output(), ibm_is_linear(), ibm_is_rowwise(), ibm_jacobian(), ibm_linear(), ibm_n(), ibm_n_output(), ibm_names(), ibm_simplify(), ibm_values()

---

ibm_eval2 *Evaluate a mapper and its Jacobian*

Description

Implementations must return a list with elements offset and jacobian. The input contents must be in a format accepted by ibm_jacobian() for the mapper.

Usage

```
ibm_eval2(mapper, input, state = NULL, ...)

## Default S3 method:
ibm_eval2(mapper, input, state, ...)

## S3 method for class 'bm_pipe'
ibm_eval2(mapper, input, state = NULL, ...)
```

Arguments

mapper	A mapper S3 object, inheriting from bru_mapper.
input	Data input for the mapper.
state	A vector of latent state values for the mapping, of length ibm_n(mapper, inla_f = FALSE)
...	Arguments passed on to other methods

Methods (by class)

- ibm_eval2(default): Calls jacobian <- ibm_jacobian(...) and offset <- ibm_eval(..., jacobian = jacobian) and returns a list with elements offset and jacobian, as needed by ibm_linear.default() and similar methods. Mapper classes can implement their own ibm_eval2 method if joint construction of evaluation and Jacobian is more efficient than separate or sequential construction.

See Also

Other mapper methods: bru_mapper_generics, ibm_eval(), ibm_inla_subset(), ibm_invalid_output(), ibm_is_linear(), ibm_is_rowwise(), ibm_jacobian(), ibm_linear(), ibm_n(), ibm_n_output(), ibm_names(), ibm_simplify(), ibm_values()

---

ibm_inla_subset *Find index subset of INLA visible states*

Description

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.

Usage

```
ibm_inla_subset(mapper, ...)

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

Arguments

mapper	A mapper S3 object, inheriting from bru_mapper.
...	Arguments passed on to other methods

Methods (by class)

- ibm_inla_subset(default): Uses the [ibm_values()] output to construct the inla subset indexing as the data.frame values.

See Also

Other mapper methods: bru_mapper_generics, ibm_eval(), ibm_eval2(), ibm_invalid_output(), ibm_is_linear(), ibm_is_rowwise(), ibm_jacobian(), ibm_linear(), ibm_n(), ibm_n_output(), ibm_names(), ibm_simplify(), ibm_values()

---

ibm_input *Interface between* bru_input *and* bru_mapper

Description

Associate bru_input objects with bru_mapper objects.

Usage

```
ibm_input_set(mapper, input)

ibm_input_new(mapper, ...)

ibm_input_available(mapper)

ibm_input_get(mapper)

bm_autodetect()
```

Arguments

mapper	A bru_mapper object.
input	A bru_input object, or NULL to remove any existing input. Alternatively, an existing bru_mapper object with or without associated input can be provided, in which case the input from that mapper is copied.
...	Passed on to new_bru_input().

Functions

- `ibm_input_set()`: Add an existing `bru_input` to a `bru_mapper`.
- `ibm_input_new()`: Create and add a `bru_input` to a `bru_mapper`.
- `ibm_input_available()`: Check if a `bru_input` is associated with a `bru_mapper`.
- `ibm_input_get()`: Get the `bru_input` associated with a `bru_mapper`.
- `bm_autodetect()`: Create a `bru_mapper` placeholder object of class `bm_autodetect`. The main purpose of this class is to attach bru_input information to it, which is later used to determine a suitable 'real' mapper type.

See Also

`bru_input()`

Examples

```
(m <- bm_autodetect())
ibm_input_available(m)
(m <- ibm_input_new(m, cos(x)))
ibm_input_available(m)
ibm_input_set(bm_linear(), m)
```

---

ibm_invalid_output *Detect invalid input to a mapper*

Description

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.

Usage

```
ibm_invalid_output(mapper, input, state, ...)

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

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

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

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

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

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

Arguments

mapper	A mapper S3 object, inheriting from bru_mapper.
input	Data input for the mapper.
state	A vector of latent state values for the mapping, of length ibm_n(mapper, inla_f = FALSE)
...	Arguments passed on to other methods
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 bm_collect.
multi	logical; If TRUE (or positive), recurse one level into sub-mappers

Methods (by class)

- ibm_invalid_output(default): Returns an all-FALSE logical vector.
- ibm_invalid_output(bm_index): Returns TRUE out-of-range input values. Returns FALSE for in-range and NA input values (to support NA giving zero effect).
- ibm_invalid_output(bm_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.bm_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(bm_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.bm_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.
- ibm_invalid_output(bm_repeat): Passes on the input to the corresponding method.
- ibm_invalid_output(bm_sum): Accepts a list with named entries, or a list with unnamed but ordered elements. The names must match the sub-mappers, see ibm_names.bm_sum(). 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.

See Also

Other mapper methods: bru_mapper_generics, ibm_eval(), ibm_eval2(), ibm_inla_subset(), ibm_is_linear(), ibm_is_rowwise(), ibm_jacobian(), ibm_linear(), ibm_n(), ibm_n_output(), ibm_names(), ibm_simplify(), ibm_values()

`ibm_is_linear` *Check if a mapper is linear/affine*

Description

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/affine.

Usage

```
ibm_is_linear(mapper, ...)

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

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

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

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

Arguments

mapper	A mapper S3 object, inheriting from bru_mapper.
...	Arguments passed on to other methods
multi	logical; If TRUE (or positive), recurse one level into sub-mappers
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 bm_collect.

Methods (by class)

- ibm_is_linear(default): Returns logical is_linear from the mapper object if it exists, and otherwise TRUE.

See Also

Other mapper methods: bru_mapper_generics, ibm_eval(), ibm_eval2(), ibm_inla_subset(), ibm_invalid_output(), ibm_is_rowwise(), ibm_jacobian(), ibm_linear(), ibm_n(), ibm_n_output(), ibm_names(), ibm_simplify(), ibm_values()

---

ibm_is_rowwise *Check if a mapper is rowwise*

Description

Implementations must return TRUE or FALSE. If TRUE (returned by the default method unless the mapper contains an is_rowwise variable), users of the mapper may assume the mapper uses its inputs in "rowwise" manner, so that blockwise evaluation is always possible.

Usage

```
ibm_is_rowwise(mapper, ...)

## Default S3 method:
ibm_is_rowwise(mapper, ...)
```

Arguments

mapper	A mapper S3 object, inheriting from bru_mapper.
...	Arguments passed on to other methods

Methods (by class)

- ibm_is_rowwise(default): Returns logical is_rowwise from the mapper object if it exists, and otherwise TRUE.

See Also

Other mapper methods: bru_mapper_generics, ibm_eval(), ibm_eval2(), ibm_inla_subset(), ibm_invalid_output(), ibm_is_linear(), ibm_jacobian(), ibm_linear(), ibm_n(), ibm_n_output(), ibm_names(), ibm_simplify(), ibm_values()

ibm_jacobian *Jacobian of a mapper*

Description

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.

Usage

```
ibm_jacobian(mapper, input, state = NULL, inla_f = FALSE, ...)

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

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

## S3 method for class 'bm_fm_mesh_1d'
ibm_jacobian(mapper, input, ...)

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

## S3 method for class 'bm_taylor'
ibm_jacobian(mapper, ..., multi = FALSE)

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

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

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

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

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

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

## S3 method for class 'bm_aggregate'
```

```
ibm_jacobian(mapper, input, state = NULL, ...)

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

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

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

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

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

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

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

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

## S3 method for class 'bm_repeat'
ibm_jacobian(
  mapper,
  input,
  state = NULL,
  inla_f = FALSE,
```

```
  multi = FALSE,
  ...,
  sub_lin = NULL
)

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

Arguments

mapper	A mapper S3 object, inheriting from bru_mapper.
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 bm_collect.
...	Arguments passed on to other methods
multi	logical; If TRUE (or positive), recurse one level into sub-mappers
reverse	logical; control bm_marginal evaluation. Default FALSE. When TRUE, reverses the direction of the mapping, see details for marginal mappers.
sub_A	Internal; precomputed Jacobian matrices.
sub_lin	Internal, optional pre-computed sub-mapper information

Methods (by class)

- ibm_jacobian(default): Mapper classes must implement their own ibm_jacobian method.
- ibm_jacobian(bm_fmesher): Returns the fmesher::fm_basis() matrix of the mesh being mapped.
- ibm_jacobian(bm_fm_mesh_1d): Returns the fmesher::fm_basis() matrix of the mesh being mapped.
- ibm_jacobian(bm_matrix): Accepts input as a matrix, Matrix, Spatial, or sfc_POINT object.
- ibm_jacobian(bm_shift): input NULL values are interpreted as no shift.

- `ibm_jacobian(bm_scale)`: `input` NULL values are interpreted as no scaling.
- `ibm_jacobian(bm_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.
- `ibm_jacobian(bm_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(bm_logitaverage)`: `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(bm_marginal)`: Non-NULL `input` values are interpreted as a parameter list for `qfun`, overriding that of the mapper itself.
- `ibm_jacobian(bm_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.bm_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(bm_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.bm_collect()`. Each list element should take a format accepted by the corresponding sub-mapper. In case each element is a vector, the input can be given as a data.frame with named columns, a matrix with named columns, or a matrix with unnamed but ordered columns. When `inla_f=TRUE` and `hidden=TRUE` in the mapper definition, the input format should instead match that of the first, non-hidden, sub-mapper.
- `ibm_jacobian(bm_repeat)`: The input should take the format of the repeated submapper.
- `ibm_jacobian(bm_sum)`: Accepts a list with named entries, or a list with unnamed but ordered elements. The names must match the sub-mappers, see `ibm_names.bm_sum()`. 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.

See Also

Other mapper methods: `bru_mapper_generics`, `ibm_eval()`, `ibm_eval2()`, `ibm_inla_subset()`, `ibm_invalid_output()`, `ibm_is_linear()`, `ibm_is_rowwise()`, `ibm_linear()`, `ibm_n()`, `ibm_n_output()`, `ibm_names()`, `ibm_simplify()`, `ibm_values()`

`ibm_linear` *Compute a mapper linearisation*

Description

Implementations must return a bm_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.

Usage

```
ibm_linear(mapper, input, state = NULL, ...)

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

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

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

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

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

## S3 method for class 'bru_comp'
ibm_linear(mapper, input, state = NULL, ...)
```

Arguments

`mapper`	A mapper S3 object, inheriting from `bru_mapper`.
`input`	Data input for the mapper.
`state`	A vector of latent state values for the mapping, of length `ibm_n(mapper, inla_f = FALSE)`
`...`	Arguments passed on to other methods
`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 `bm_collect`.

Value

A bm_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)
```

Methods (by class)

- `ibm_linear(default)`: Calls `ibm_eval2()` and returns a bm_taylor object.

See Also

Other mapper methods: `bru_mapper_generics`, `ibm_eval()`, `ibm_eval2()`, `ibm_inla_subset()`, `ibm_invalid_output()`, `ibm_is_linear()`, `ibm_is_rowwise()`, `ibm_jacobian()`, `ibm_n()`, `ibm_n_output()`, `ibm_names()`, `ibm_simplify()`, `ibm_values()`

---

ibm_n *Size of the latent vector of a mapping*

Description

Implementations must return the size of the latent vector being mapped to.

Usage

```
ibm_n(mapper, inla_f = FALSE, ...)

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

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

## S3 method for class 'bm_fm_mesh_1d'
ibm_n(mapper, ...)

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

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

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

## S3 method for class 'bm_factor'
```

```
ibm_n(mapper, ...)

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

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

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

## S3 method for class 'bm_aggregate'
ibm_n(mapper, ..., input = NULL, state = NULL, n_state = NULL)

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

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

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

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

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

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

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

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

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 bm_collect.
...	Arguments passed on to other methods

multi	logical; If TRUE (or positive), recurse one level into sub-mappers
state	A vector of latent state values for the mapping, of length ibm_n(mapper, inla_f = FALSE)
n_state	integer giving the length of the state vector for mappers that have state dependent output size.
input	Data input for the mapper.

Methods (by class)

- 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(bm_fmesher): Returns the fmesher::fm_dof() value of the mesh being mapped.
- ibm_n(bm_fm_mesh_1d): Returns the fmesher::fm_dof() value of the mesh being mapped.
- ibm_n(bm_linear): Returns 1L
- ibm_n(bm_matrix): Returns the number of columns in the matrix mapper.
- ibm_n(bm_factor): Returns the number of levels when factor_mapping is "full", and the number of levels minus one if factor_mapping is "contrast".

See Also

Other mapper methods: bru_mapper_generics, ibm_eval(), ibm_eval2(), ibm_inla_subset(), ibm_invalid_output(), ibm_is_linear(), ibm_is_rowwise(), ibm_jacobian(), ibm_linear(), ibm_n_output(), ibm_names(), ibm_simplify(), ibm_values()

---

ibm_n_output *Output size of a mapping*

Description

Implementations must return an integer denoting the mapper output length. The default implementation returns NROW(input). Mappers such as bm_multi and bm_collect, that can accept list() inputs require their own method implementations.

Usage

```
ibm_n_output(mapper, input, state = NULL, inla_f = FALSE, ...)

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

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

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

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

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

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

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

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

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

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

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

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

Arguments

mapper	A mapper S3 object, inheriting from bru_mapper.
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 bm_collect.
...	Arguments passed on to other methods
n_state	integer giving the length of the state vector for mappers that have state dependent output size.
multi	logical; If TRUE (or positive), recurse one level into sub-mappers

Methods (by class)

- ibm_n_output(default): Returns NROW(input)

See Also

Other mapper methods: bru_mapper_generics, ibm_eval(), ibm_eval2(), ibm_inla_subset(), ibm_invalid_output(), ibm_is_linear(), ibm_is_rowwise(), ibm_jacobian(), ibm_linear(), ibm_n(), ibm_names(), ibm_simplify(), ibm_values()

`ibm_names` *Names of submapper*

Description

Implementations must return a character vector of sub-mapper names, or NULL. Intended for providing information about multi-mappers and mapper collections.

Usage

```
ibm_names(mapper)

ibm_names(mapper) <- value

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

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

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

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

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

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

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

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

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

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

Arguments

mapper	A mapper S3 object, inheriting from bru_mapper.
value	a character vector of up to the same length as the number of mappers in the multi-mapper x
...	Arguments passed on to other methods

Methods (by class)

- ibm_names(default): Returns NULL
- ibm_names(bm_multi): Returns the names from the sub-mappers list
- ibm_names(bm_collect): Returns the names from the sub-mappers list
- ibm_names(bm_sum): Returns the names from the sub-mappers list

Functions

- ibm_names(mapper) <- value: Set mapper names.

See Also

Other mapper methods: bru_mapper_generics, ibm_eval(), ibm_eval2(), ibm_inla_subset(), ibm_invalid_output(), ibm_is_linear(), ibm_is_rowwise(), ibm_jacobian(), ibm_linear(), ibm_n(), ibm_n_output(), ibm_simplify(), ibm_values()

Examples

```
# ibm_names
mapper <- bm_multi(list(A = bm_index(2), B = bm_index(2)))
ibm_names(mapper)
ibm_names(mapper) <- c("new", "names")
ibm_names(mapper)
```

---

ibm_simplify *Simplify a mapper*

Description

Implementations must return a bru_mapper object. The default method returns the result of ibm_linear() for linear/affine mappers, and the original mapper for non-linear mappers.

Usage

```
ibm_simplify(mapper, input = NULL, state = NULL, ...)

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

## S3 method for class 'bm_pipe'
ibm_simplify(
  mapper,
  input = NULL,
  state = NULL,
  inla_f = FALSE,
  ...,
  n_state = NULL
)
```

Arguments

mapper	A mapper S3 object, inheriting from bru_mapper.
input	Data input for the mapper.
state	A vector of latent state values for the mapping, of length ibm_n(mapper, inla_f = FALSE)
...	Arguments passed on to other methods
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 bm_collect.
n_state	integer giving the length of the state vector for mappers that have state dependent output size.

Value

The original mapper is returned for non-linear mappers, and the output of ibm_linear() is returned for linear mappers.

Methods (by class)

- ibm_simplify(default): Calls ibm_linear() for linear mappers, and returns the original mapper for non-linear mappers.
- ibm_simplify(bm_pipe): Constructs a simplified pipe mapper. For fully linear pipes, calls ibm_linear(). For partially non-linear pipes, replaces each sequence of linear mappers with a single bm_taylor() mapper, while keeping the full list of original mapper names, allowing the original input structure to be used also with the simplified mappers, since the taylor mappers are not dependent on inputs.

See Also

Other mapper methods: bru_mapper_generics, ibm_eval(), ibm_eval2(), ibm_inla_subset(), ibm_invalid_output(), ibm_is_linear(), ibm_is_rowwise(), ibm_jacobian(), ibm_linear(), ibm_n(), ibm_n_output(), ibm_names(), ibm_values()

`ibm_values` *Value vector for a mapping*

Description

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 `bm_multi`, that returns a multi-column data frame if `multi=TRUE`.

Usage

```
ibm_values(mapper, inla_f = FALSE, ...)

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

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

## S3 method for class 'bm_fm_mesh_1d'
ibm_values(mapper, ...)

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

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

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

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

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

## S3 method for class 'bm_shift'
ibm_values(mapper, ..., state = NULL, n_state = NULL)

## S3 method for class 'bm_scale'
```

```
ibm_values(mapper, ..., state = NULL, n_state = NULL)

## S3 method for class 'bm_aggregate'
ibm_values(mapper, ..., state = NULL, n_state = NULL)

## S3 method for class 'bm_marginal'
ibm_values(mapper, ..., state = NULL, n_state = NULL)

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

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

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

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

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

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

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 bm_collect.
...	Arguments passed on to other methods
multi	logical; If TRUE (or positive), recurse one level into sub-mappers
state	A vector of latent state values for the mapping, of length ibm_n(mapper, inla_f = FALSE)
n_state	integer giving the length of the state vector for mappers that have state dependent output size.

Methods (by class)

- 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_values(bm_fmesher): Returns an index vector for the mesh basis functions.

- `ibm_values(bm_fm_mesh_1d)`: Returns an index vector into the basis functions for an indexed mapper. Otherwise, the `mid` values if present in the mesh being mapped, and otherwise returns the `loc` values of the mesh.
- `ibm_values(bm_linear)`: Returns `1.0`
- `ibm_values(bm_matrix)`: For integer labels, the vector of labels. For character labels, the labels as a factor variable.
- `ibm_values(bm_factor)`: Returns the factor levels (minus the first level for `factor_mapping` "contrast"), or an integer vector (if `indexed = TRUE` in `bm_factor()`).

See Also

Other mapper methods: `bru_mapper_generics`, `ibm_eval()`, `ibm_eval2()`, `ibm_inla_subset()`, `ibm_invalid_output()`, `ibm_is_linear()`, `ibm_is_rowwise()`, `ibm_jacobian()`, `ibm_linear()`, `ibm_n()`, `ibm_n_output()`, `ibm_names()`, `ibm_simplify()`

`lgcp` *Log Gaussian Cox process (LGCP) inference using INLA*

Description

This function performs inference on a LGCP observed via points residing possibly multiple dimensions. These dimensions are defined via the left hand side of the formula provided via the model parameter. The left hand side determines the intensity function that is assumed to drive the LGCP. This may include effects that lead to a thinning (filtering) of the point process. By default, the log intensity is assumed to be a linear combination of the effects defined by the formula's RHS.

More sophisticated models, e.g. non-linear thinning, can be achieved by using the predictor argument. The latter requires multiple runs of INLA for improving the required approximation of the predictor. In many applications the LGCP is only observed through subsets of the dimensions the process is living in. For example, spatial point realizations may only be known in sub-areas of the modelled space. These observed subsets of the LGCP domain are called samplers and can be provided via the respective parameter. If samplers is NULL it is assumed that all of the LGCP's dimensions have been observed completely.

Usage

```
lgcp(
  components,
  data,
  domain = NULL,
  samplers = NULL,
  ips = NULL,
  formula = . ~ .,
  E = NULL,
  weights = NULL,
  ...,
  options = list(),
  .envir = parent.frame()
)
```

Arguments

components	Latent component definitions, either as a bru_comp_list() object, or a formula-like specification. Also used to define a default linear additive predictor. See bru_comp() for details.
data	Predictor expression-specific data, as a data.frame, tibble, or sf. Since 2.12.0.9023, deprecated support for SpatialPoints[DataFrame] objects.
domain, samplers, ips	Arguments used for family="cp" and aggregate=. domain Named list of domain definitions, see fmesher::fm_int(). samplers Integration domain for family="cp" or subdomains for aggregate=, see fmesher::fm_int(). ips Integration points. Defaults to fmesher::fm_int(domain, samplers). If explicitly given, overrides domain and samplers. fmesher::new_fm_int() (from fmesher 0.5.0.9013) can be used for manually constructed integration schemes.
formula	a formula where the right hand side is a general R expression defines the predictor used in the model.
E	Exposure/effort parameter for family = 'poisson' passed on to INLA::inla. Special case if family is 'cp': rescale all integration weights by a scalar E. For sampler specific reweighting/effort, use a weight column in the samplers object instead, see fmesher::fm_int(). Default taken from options$E, 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. For family = "cp", the weights are applied as sum(weights * eta) in the point location contribution part of the log-likelihood, where eta is the linear predictor, and do not affect the integration part of the likelihood. This can be used to implement approximative methods for point location uncertainty.
...	Further arguments passed on to bru_obs().
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, weights, and scale) if not found in response_data or data. Defaults to the calling environment.

Details

The E and weights arguments are evaluated in the data context, like for bru_obs().

Value

An bru() object

Examples

```
if (bru_safe_inla() &&
  require(ggplot2, quietly = TRUE) &&
  require(fmesher, quietly = TRUE) &&
  requireNamespace("sn", quietly = TRUE)) {
  # Load the Gorilla data
  data <- gorillas_sf

  # Plot the Gorilla nests, the mesh and the survey boundary
  ggplot() +
    fmesher::geom_fm(data = data$mesh) +
    gg(data$boundary, fill = "blue", alpha = 0.2) +
    gg(data$nests, col = "red", alpha = 0.2)

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

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

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

  # Predict the spatial intensity surface
  lambda <- predict(
    fit,
    fmesher::fm_pixels(data$mesh, mask = data$boundary),
    ~ exp(field + Intercept)
  )

  # Plot the intensity
  ggplot() +
    gg(lambda, geom = "tile") +
    gg(data$nests, col = "red", alpha = 0.2)
}
```

`mexdolphin_sf` *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, with the parts stored in sf format. 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

```
mexdolphin_sf

mexdolphin_sp()
```

Format

A list of objects:

points: An sf object containing the locations of detected dolphin groups, with their size as an attribute.

samplers: An sf object containing the transect lines that were surveyed.

mesh: An fm_mesh_2d object containing a Delaunay triangulation mesh (a type of discretization of continuous space) covering the survey region.

ppoly: An sf object defining the boundary of the survey region.

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

Functions

- mexdolphin_sp(): Convert mexdolphin_sf to sp format. Replaces the old mexdolphin dataset.

Source

Library dsm.

References

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

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

Examples

```
if (require("ggplot2", quietly = TRUE)) {
  data(mexdolphin_sf, package = "inlabru", envir = environment())
  ggplot() +
    gg(mexdolphin_sf$mesh) +
    gg(mexdolphin_sf$ppoly, color = "blue", alpha = 0, linewidth = 1) +
    gg(mexdolphin_sf$samplers) +
    gg(mexdolphin_sf$points, aes(size = size), color = "red") +
    scale_size_area()

  ggplot() +
    gg(mexdolphin_sf$mesh,
      color = mexdolphin_sf$lambda,
      mask = mexdolphin_sf$ppoly
    )
}


if (require("ggplot2", quietly = TRUE) &&
  require("sp", quietly = TRUE)) {
  mexdolphin <- mexdolphin_sp()
  ggplot() +
    gg(mexdolphin$mesh) +
    gg(mexdolphin$ppoly, color = "blue") +
    gg(mexdolphin$samplers) +
    gg(mexdolphin$points, aes(size = size), color = "red") +
    scale_size_area() +
    coord_equal()

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

`mrsea` *Marine renewables strategic environmental assessment*

Description

Data imported from package MRSea, see https://www.creem.st-andrews.ac.uk/software/

Usage

```
mrsea
```

Format

A list of objects:

`points` A `sf` object containing the locations of XXXXX.

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

`mesh` An `fm_mesh_2d` object containing a Delaunay triangulation mesh (a type of discretization of continuous space) covering the survey region.

`boundary` An `sf` object defining the boundary polygon of the survey region.

`covar` An `sf` containing sea depth estimates.

Source

Library `MRSea`.

References

NONE YET

Examples

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

`multiplot` *Multiple ggplots on a page.*

Description

[Deprecated] in favour of the `patchwork` package; see the example below.

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

```
if (require("ggplot2", quietly = TRUE)) {
  df <- data.frame(x = 1:10, y = cos(1:10), z = sin(1:10))
  pl1 <- ggplot(data = df) +
    geom_line(mapping = aes(x, y), color = "red")
  pl2 <- ggplot(data = df) +
    geom_line(mapping = aes(x, z), color = "blue")
  pl3 <- ggplot(data = df) +
    geom_path(mapping = aes(y, z), color = "magenta")

  multiplot(
    pl1, pl2, pl3,
    layout = rbind(c(1, 2), c(3, 3))
  )

  # Recommended alternative using the patchwork package:
  if (require("patchwork")) {
    (pl1 | pl2) / pl3
  }
}
```

---

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

Description

From version 2.11.0, plot.bru(x, ...) calls plot.inla(x, ...) from the INLA package, unless the first argument after x is a character, in which case the pre-2.11.0 behaviour is used, calling plotmarginal.inla(x, ...) instead.

Requires the ggplot2 package.

Usage

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

plotmarginal.inla(
  result,
  varname = NULL,
  index = NULL,
  link = function(x) {
     x
 },
  add = FALSE,
  ggp = TRUE,
  lwd = 3,
  ...
)
```

Arguments

x	a fitted bru() model.
...	Options passed on to other methods.
result	an inla or bru result object
varname	character; name of the variable to plot
index	integer; index of the random effect to plot
link	function; link function to apply to the variable
add	logical; if TRUE, add to an existing plot
ggp	logical; unused
lwd	numeric; line width

Examples

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

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

## End(Not run)
```

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

```
# Some features require the raster package
if (bru_safe_sp() &&
  require("sp") &&
  require("raster", quietly = TRUE) &&
  require("ggplot2", quietly = TRUE) &&
  bru_safe_terra(quietly = TRUE) &&
  require("sf", quietly = TRUE)) {
  gorillas <- gorillas_sp()
```

```
  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

```
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

```
# Some features require the raster package
if (bru_safe_sp() &&
  require("sp") &&
  require("raster", quietly = TRUE) &&
  require("ggplot2", quietly = TRUE) &&
  bru_safe_terra(quietly = TRUE) &&
  require("sf", quietly = TRUE) &&
  require("patchwork", quietly = TRUE)) {
  gorillas <- gorillas_sp()
  plotpts <- plotsample(gorillas$nests, gorillas$boundary,
    x.ppn = 0.4, y.ppn = 0.4, nx = 5, ny = 5
  )
  p1 <- ggplot() +
    gg(plotpts$plots) +
```

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

`Poisson1_1D` *1-Dimensional Homogeneous Poisson example.*

Description

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

Usage

```
data(Poisson1_1D)
```

Format

The data contain the following R objects:

`lambda1_1D` A function defining the intensity function of a nonhomogeneous Poisson process. Note that this function is only defined on the interval (0,55).

`E_nc1` The expected counts of the gridded data.

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

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

- `x` The grid cell midpoint.
- `count` The number of detections in the cell.
- `exposure` The width of the cell.

Examples

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

```
    geom_point(
      data = countdata1, aes(x = x), y = 0, shape = "+",
      col = "blue", cex = 4
    ) +
    xlab(expression(bold(s))) +
    ylab("count")
}
```

Poisson2_1D *1-Dimensional NonHomogeneous Poisson example.*

Description

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

Usage

```
data(Poisson2_1D)
```

Format

The data contain the following R objects:

`lambda2_1D`**:** A function defining the intensity function of a nonhomogeneous Poisson process. Note that this function is only defined on the interval (0,55).

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

`E_nc2` The expected counts of the gridded data.

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

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

- `x` The grid cell midpoint.
- `count` The number of detections in the cell.
- `exposure` The width of the cell.

Examples

```
if (require("ggplot2", quietly = TRUE) &&
  require("patchwork", 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.out = 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))))
  (p1 / p2)
}
```

`Poisson3_1D` *1-Dimensional NonHomogeneous Poisson example.*

Description

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

Usage

```
data(Poisson3_1D)
```

Format

The data contain the following R objects:

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

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

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

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

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

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

- `x` The grid cell midpoint.
- `count` The number of detections in the cell.
- `exposure` The width of the cell.

Examples

```
if (require("ggplot2", quietly = TRUE) &&
  require("patchwork", 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.out = 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))))
  (p1a / p2a)

  # Then the plots for the 20-bin case:
  p1a <- ggplot(countdata3b) +
    geom_point(data = countdata3b, aes(x = x, y = count), col = "blue") +
    ylim(0, max(countdata3b$count, E_nc3b)) +
    geom_point(
      data = countdata3b, aes(x = x), y = 0, shape = "+",
      col = "blue", cex = 4
    ) +
    geom_point(
      data = data.frame(x = countdata3b$x, y = E_nc3b),
      aes(x = x), y = E_nc3b, shape = "_", cex = 5
    ) +
    xlab(expression(bold(s))) +
    ylab("count")
  ss <- seq(0, 55, length.out = 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))))
  (p1a / p2a)
}
```

---

predict.bru *Prediction from fitted bru model*

Description

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

Usage

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

Arguments

object	An object obtained by calling bru() or lgcp().
newdata	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 "1:1:1"

used — Either NULL or a bru_used() object. Default, NULL, uses auto-detection of used variables in the formula.

drop — logical; If drop=FALSE, and the prediction summary has the same number of rows as newdata, then the output is a joined object. Default FALSE.

... — Additional arguments passed on to inla.posterior.sample()

include, exclude — **[Deprecated]** If auto-detection of used variables fails, use used instead.

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 bru_comp_eval()).

For "iid" models with mapper = bm_index(n), rnorm() is used to generate new realisations for indices greater than n, if accessed via <name>_eval(...).

Value

a data.frame, sf, or Spatial* object with predicted mean values and other summary statistics attached. Non-S4 object outputs have the class "bru_prediction" added at the front of the class list.

Examples

```
if (bru_safe_inla() &&
  requireNamespace("sn", quietly = TRUE) &&
  require("ggplot2", quietly = TRUE) &&
  bru_safe_terra(quietly = TRUE) &&
  require("sf", quietly = TRUE)) {
  # Load the Gorilla data

  gorillas <- gorillas_sf

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

  ggplot() +
    gg(gorillas$mesh) +
    gg(gorillas$nests) +
    gg(gorillas$boundary, alpha = 0.1)
```

```
# 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 <- geometry ~ field(geometry, model = matern) + Intercept(1)

# Fit the model, with "eb" instead of full Bayes
fit <- lgcp(
  cmp,
  data = gorillas$nests,
  samplers = gorillas$boundary,
  domain = list(geometry = 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 'field'.
# In order to predict 'field' 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 sf (and legacy sp/Spatial) objects.
# For instance, we might want to predict 'field' at the locations of the
# mesh vertices. Using

vrt <- fmesher::fm_vertices(gorillas$mesh, format = "sf")

# we obtain these vertices as an sf object with POINT geometries
```

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

  # Predicting 'field' at these locations works as follows

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

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

  class(field)
  head(vrt)
  head(field)

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

  ggplot() +
    gg(gorillas$mesh) +
    gg(field, aes(color = mean), size = 2)

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

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

  # Alternatively, we can predict the spatial field at a grid of locations,
  # e.g. a sf object with a grid of points covering the relevant part of mesh

  pxl <- fmesher::fm_pixels(gorillas$mesh,
    format = "sf",
    mask = gorillas$boundary
  )
  field2 <- predict(fit, pxl, ~field)

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

  head(field2)
  ggplot() +
    gg(gorillas$boundary) +
    gg(data = field2, geom = "tile")
}
```

---

robins_subset *robins_subset*

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

```
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

Meehan, T.D., Michel, N.L., and Rue, H. 2019. Spatial modeling of Audubon Christmas Bird Counts reveals fine-scale patterns and drivers of relative abundance trends. Ecosphere, 10(4), p.e02707.

Examples

```
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")
}
```

`sample.lgcp` *Sample from an inhomogeneous Poisson process*

Description

This function provides point samples from one- and two-dimensional inhomogeneous Poisson processes. The log intensity has to be provided via its values at the nodes of an `fm_mesh_1d` or `fm_mesh_2d` object. In between mesh nodes the log intensity is assumed to be linear.

Usage

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

Arguments

`mesh`	An fmesher::fm_mesh_1d or fmesher::fm_mesh_2d object
`loglambda`	vector or matrix; A vector of log intensities at the mesh vertices (for higher order basis functions, e.g. for `fm_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 `fm_mesh_2d` 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), or sf (2D flat and 3D spherical surface cases, or 2D/2.5D 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.

For the old (pre version 2.13.0.9034) sp output format, use as(ret, "Spatial") or sf::as_Spatial(ret).

Author(s)

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

Examples

```
# The INLA package is required
if (bru_safe_inla()) {
  vertices <- seq(0, 3, by = 0.1)
  mesh <- fmesher::fm_mesh_1d(vertices)
  loglambda <- 5 - 0.5 * vertices
  pts <- sample.lgcp(mesh, loglambda)
  pts$y <- 0
  plot(vertices, exp(loglambda), type = "l", ylim = c(0, 150))
  points(pts, pch = "|")
}



# The INLA package is required
if (bru_safe_inla() &&
  require(ggplot2, quietly = TRUE) &&
  bru_safe_terra(quietly = TRUE) &&
  require("sf", quietly = TRUE)) {
  gorillas <- gorillas_sf
  pts <- sample.lgcp(gorillas$mesh,
```

```
  loglambda = 1.5,
  samplers = gorillas$boundary
)
ggplot() +
  gg(gorillas$mesh) +
  gg(pts)

pts <- sample.lgcp(gorillas$mesh,
  loglambda = base::scale(gorillas$mesh$loc) * 2,
  samplers = gorillas$boundary,
  ignore.CRS = TRUE
)
ggplot() +
  gg(gorillas$mesh) +
  gg(pts, aes(color = as.factor(sample))) +
  labs(color = "Sample")
}
```

---

shrimp *Blue and red shrimp in the Western Mediterranean Sea*

Description

Blue and red shrimp in the Western Mediterranean Sea.

Usage

```
data(shrimp)
```

Format

A list of objects:

`hauls`: An `sf` object containing haul locations

`mesh`: An `fm_mesh_2d` 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 (in metres) of the fishery haul.

Source

Pennino, Maria Grazia. Personal communication.

References

Pennino, M. G., Paradinas, I., Munoz, F., Illian, J.,Quilez-Lopez, A., Bellido, J.M., Conesa, D. Accounting for preferential sampling in species distribution models. Ecology and Evolution, 9(1), p653-663, 2019 doi:10.1002/ece3.4789

Examples

```
if (require(ggplot2, quietly = TRUE)) {
  data(shrimp, package = "inlabru", envir = environment())
  ggplot() +
    fmesher::geom_fm(data = shrimp$mesh) +
    gg(shrimp$hauls, aes(col = catch)) +
    coord_sf(datum = fmesher::fm_crs(shrimp$hauls))
}
```

`spde.posterior` *Posteriors 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", quantile = 0.95)
```

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".
`quantile`	The target credible probability. Default 0.95.

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.out = 20)
  mesh1D <- fmesher::fm_mesh_1d(x, boundary = "free")
  constrained_matern <- INLA::inla.spde2.matern(mesh1D, constr = TRUE)
  mdl <- x ~ spde1D(x, model = constrained_matern) + Intercept(1)
  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)
}
```

---

summary.bru_obs *Summary and print methods for observation models*

Description

Summary and print methods for observation models

Usage

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

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

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

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

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

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

Arguments

object	Object to operate on
verbose	logical; If TRUE, include more details of the component definitions. If FALSE, only show basic component definition information. Default: TRUE
...	Arguments passed on to other summary methods
x	Object to be printed

See Also

bru_obs()

Examples

```
obs <- bru_obs(y ~ ., data = data.frame(y = rnorm(10)))
summary(obs)
print(obs)
```

---

summary.bru_options *Print inlabru options*

Description

Print inlabru options

Usage

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

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

Arguments

object	A bru_options object to be summarised
legend	logical; If TRUE, include explanatory text, Default: TRUE
include_global	logical; If TRUE, include global override options
include_default	logical; If TRUE, include default options
...	Further parameters, currently ignored
x	A summary_bru_options object to be printed

Examples

```
if (interactive()) {
  options <- bru_options(verbose = TRUE)

  # Don't print options only set in default:
  print(options, include_default = FALSE)

  # Only include options set in the object:
  print(options, include_default = FALSE, include_global = FALSE)
}
```

tidy.bru *Tidy a bru model fit*

Description

Tidy a bru model fit

Usage

```
## S3 method for class 'bru'
tidy(x, effects = "fixed", ...)
```

Arguments

x	A fitted bru object.
effects	"fixed" (default) or "hyperpar".
...	Unused.

Value

A tibble with one row per term.

toygroups *Simulated 1D animal group locations and group sizes*

Description

This data set serves to teach the concept of modelling species that gather in groups and where the grouping behaviour depends on space.

Usage

```
data(toygroups)
```

Format

The data are a list that contains these elements:

groups: A data.frame of group locations x and size size

df.size: IGNORE THIS

df.intensity: A data.frame with Poisson process intensity d.lambda at locations x

df.rate: A data.frame the locations x and associated rate which parameterized the exponential distribution from which the group sizes were drawn.

Examples

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

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

  # The data set is a simulation of animal groups residing in a 1D space.
  # Their locations in x-space are sampled from a Cox process with
  # intensity

  ggplot(toygroups$df.intensity) +
    geom_line(aes(x = x, y = g.lambda))

  # Adding the simulated group locations to this plot we obtain

  ggplot(toygroups$df.intensity) +
    geom_line(aes(x = x, y = g.lambda)) +
    geom_point(data = toygroups$groups, aes(x, y = 0), pch = "|")

  # Each group has a size mark attached to it.
  # These group sizes are sampled from an exponential distribution
  # for which the rate parameter depends on the x-coordinate

  ggplot(toygroups$groups) +
    geom_point(aes(x = x, y = size))

  ggplot(toygroups$df.rate) +
    geom_line(aes(x, rate))
}
```

---

toypoints *Simulated 2D point process data*

Description

This data set serves as an example for basic inlabru.

Usage

```
data(toypoints)
```

Format

The data are a list that contains these elements:

`points` An `sf` object of point locations and and `z` measurements

`mesh` An `fm_mesh_2d` object

`boundary` An `sf` polygon denting the region of interest

`pred_locs` A `sf` object with prediction point locations

Examples

```
if (require("ggplot2")) {
  ggplot() +
    fmesher::geom_fm(data = toypoints$mesh, alpha = 0) +
    geom_sf(data = toypoints$boundary, fill = "blue", alpha = 0.1) +
    geom_sf(data = toypoints$points, aes(color = z)) +
    scale_color_viridis_c()
}
```

Index