Package ‘inlabru’

February 16, 2020

Type Package

Title Spatial Inference using Integrated Nested Laplace Approximation

Version 2.1.13

URL http://www.inlabru.org,

BugReports https://github.com/fbachl/inlabru/issues


License GPL (>= 2)

Imports r gdal, rgeos, utils, Matrix

Suggests testthat, ggmap, rgl, sphereplot, raster, dplyr, maptools, mgcv, shiny, spatstat, spatstat.data, RColorBrewer, graphics, INLA, knitr, rmarkdown

Depends R (>= 3.3), sp, stats, methods, ggplot2,

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

RoxygenNote 6.1.1

NeedsCompilation no

Author Fabian E. Bachl [aut, cre] (Fabian Bachl wrote the main code), Finn Lindgren [aut] (<https://orcid.org/0000-0002-5833-2011>, Finn Lindgren wrote code for SPDE posterior plotting, and continued development of the main code), David L. Borchers [ctb] (David Borchers wrote code for Gorilla data import and sampling, multiplot tool), Daniel Simpson [ctb] (Daniel Simpson wrote the basic LGCP sampling method), Lindesay Scott-Howard [ctb] (Lindesay Scott-Howard provied MRSea data import code)
Maintainer  Fabian E. Bachl <bachlfab@gmail.com>
Repository  CRAN
Date/Publication  2020-02-16 22:10:09 UTC

R topics documented:

bincount ........................................ 3
bru .............................................. 4
bru.components ................................ 6
bru.options .................................... 9
cprod ........................................... 10
deltaIC .......................................... 11
devel.cvmeasure ................................ 12
generate ........................................ 14
generate.bru ................................... 15
generate.inla .................................. 17
gg ............................................... 18
gg.data.frame ................................... 19
gg.inla.mesh .................................... 21
gg.inla.mesh.1d ................................ 22
gg.matrix ....................................... 24
gg.prediction ................................... 24
gg.RasterLayer ................................ 26
gg.SpatialGridDataFrame ....................... 28
gg.SpatialLines ................................ 29
gg.SpatialPixels ................................ 31
gg.SpatialPixelsDataFrame ..................... 32
gg.SpatialPoints ................................ 33
gg.SpatialPolygons .............................. 35
globe ............................................ 37
glplot ............................................ 38
glplot.inla.mesh ................................ 39
glplot.SpatialLines ............................ 40
glplot.SpatialPoints ............................ 41
gm ................................................ 42
gmap ............................................. 43
gorillas .......................................... 44
init.tutorial .................................... 45
inlabru .......................................... 46
int .............................................. 46
ipoints ......................................... 48
lgcp ............................................. 49
like ............................................. 51
mexdolphin .................................... 53
mrsea ............................................ 54
multiplot ....................................... 55
pixels ........................................... 56
A common procedure of analyzing the distribution of 1D points is to choose a binning and plot the data’s histogram with respect to this binning. This function compares the counts that the histogram calculates to simulations from a 1D log Gaussian Cox process conditioned on the number of data samples. For each bin this results in a median number of counts as well as a confidence interval. If the LGCP is a plausible model for the observed points then most of the histogram counts (number of points within a bin) should be within the confidence intervals. Note that a proper comparison is a multiple testing problem which the function does not solve for you.

Usage

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

Arguments

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

Value

An \texttt{inla} object

Examples

```r
## Not run:
# Load a point pattern
data(Poisson2_1D)

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

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

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

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

# Plot them!
attributes(bc)$ggp

## End(Not run)
```

\textit{Convenient model fitting using (iterated) INLA}

Description

This method is a wrapper for \texttt{inla} and provides multiple enhancements.
• Easy usage of spatial covariates and automatic construction of inla projection matrices for (spatial) SPDE models. This feature is accessible via the components parameter. Practical examples on how to use spatial data by means of the components parameter can also be found by looking at the lgcp function’s documentation.

• Constructing multiple likelihoods is straight forward. See like for more information on how to provide additional likelihoods to bru using the ... parameter list.

• Support for non-linear predictors. See example below.

• Log Gaussian Cox process (LGCP) inference is available by using the cp family or (even easier) by using the lgcp function.

Usage
bru(components = y ~ Intercept, family = NULL, data = NULL, ..., options = list())

Arguments
components a formula describing the latent components. See bru.components for details.
family A string indicating the likelihood family. The default is gaussian with identity link. In addition to the likelihoods provided by inla (see inla.models()$likelihood) inlabru supports fitting Cox processes via family = "cp". The latter requires contructing a likelihood using the like function and providing it via the ... parameter list. As an alternative to bru, the lgcp function provides a convenient interface to fitting Cox processes. See details.
data A data.frame or SpatialPoints[DataFrame] object. See details.
... Additional likelihoods, each constructed by a calling like. See details.
options A list of name and value pairs that are either interpretable by bru.options or valid inla parameters.

Details
family and ... must either be parameters to like, or lhood objects constructed by like. data must either be an lhood object, a data container, or NULL. If NULL, data must be supplied through direct calls to like.

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

Author(s)
Fabian E. Bachl <bachlfab@gmail.com>
Examples

```r
if (require("INLA", quietly = TRUE)) {
  # 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, "gaussian", input.df)

  # Obtain summary
  fit$summary.fixed
}
```

```r
if (require("INLA", quietly = TRUE)) {
  # Alternatively, we can use the like() function to construct the likelihood:
  lik = like(family = "gaussian", data = input.df)
  fit <- bru(y ~ x + Intercept, lik)
  fit$summary.fixed
}
```

```r
# An important addition to the INLA methodology is bru's ability to use
# non-linear predictors. Such a predictor can be formulated via like()'s
# formula parameter. For instance
if (require("INLA", quietly = TRUE)) {
  z = 2
  input.df <- within(input.df, y <- 5 + exp(z)*x + rnorm(10, mean=0, sd=0.1))
  lik = like(family = "gaussian", data = input.df, formula = y ~ exp(z)*x + Intercept, E = 10000)
  fit <- bru(~ z + Intercept, lik)

  # Check the result (z posterior should be around 2)
  fit$summary.fixed
}
```

---

**bru.components**

**bru components**

**Description**

Similar to glm(), gam() and inla() bru uses formula objects to describe response data and latent (unknown) components of the model to be fitted. However, in addition to the syntax compatible with inla, bru components offer additional functionality which facilitates modeling.
bru.components

Usage
bru.components()

Details

bru will understand formulae describing fixed effect models just like the other methods. For instance, the formula $y \sim x$ will fit the linear combination of an effect named $x$ and an intercept to the response $y$ with respect to the likelihood family stated when calling bru. Mathematically, the linear predictor $\eta$ would be written down as

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

where:

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

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

Naming random effects

In inla, a simple random effect model would be expressed as

```r
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 $\eta = \beta \ast x + c$ but the result of fitting the model would state $x$ as the random effect’s name. bru allows to rewrite this formula in order to explicitly state the name of the random effect and the name of the associated. This is achieved by replacing $f$ with an arbitrary name that we wish to assign to the effect, e.g.

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

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

```r
predict(fit, data.frame(x=1)), ~ exp(x + Intercept).
```

On the other hand, predict may be used to only look at a transformation of the random effect $\psi$

```r
predict(fit, NULL, ~ exp(psi)).
```
Simple covariates and the map parameter

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

- $\text{formula} = y \sim f(xsquared, \text{model} = "linear")$.

In inlabru this can be achieved using two ways of using the map parameter.

- $\text{components} = y \sim \psi(\text{map} = x^2, \text{model} = "linear")$
- $\text{components} = y \sim \psi(\text{map} = \text{mySquareFun}(x), \text{model} = "linear")$
- $\text{components} = y \sim \psi(\text{map} = \text{myOtherSquareFun}, \text{model} = "linear")$.

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 directly and not within an expression. In this case, inlabru will call the function using the data provided via the data parameter. inlabru expects that the output of this function is a data.frame with "psi" being the name of the single existing column. For instance,

myOtherSquareFun = function(data) { data = data[,"x",drop = FALSE]; colnames(data) = "psi"; return(data)}

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 SpatialPixelDataFrame, SpatialPixelDataFrame or RasterLayer objects. These can be provided directly via the map parameter if the input data is a SpatialPointsDataFrame. inlabru will automatically evaluate and/or interpolate the covariate at your data locations when using code like

- $\text{components} = y \sim \psi(\text{mySpatialPixels}, \text{model} = "linear")$.

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 which maps SPDE values at the mesh vertices to values at the data locations. For this purpose, the map parameter 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. The code for this would look as follows:

- $\text{components} = y \sim \text{mySPDE(\text{map} = \text{coordinates}, \text{model} = \text{inla.spde2.matern}(...))}$.
bru.options

Author(s)

Fabian E. Bachl <<bachlfab@gmail.com>>

---

bru.options

Additional bru options

Description

Additional bru options

Usage

bru.options(mesh = NULL, run = TRUE, max.iter = 10, offset = 0, result = NULL, E = 1, Ntrials = 1, control.compute = inlabru:::inla.getOption("control.compute"), control.inla = inlabru:::inla.getOption("control.inla"), control.fixed = inlabru:::inla.getOption("control.fixed"), ...)

Arguments

mesh An inla.mesh object for spatial models without SPDE components. Mostly used for successive spatial predictions.
run If TRUE, run inference. Otherwise only return configuration needed to run inference.
max.iter maximum number of inla iterations
offset the usual inla offset. If a nonlinear formula is used, the resulting Taylor approximation constant will be added to this automatically.
result An inla object returned from previous calls of inla, bru or lgcp. This will be used as a starting point for further improvement of the approximate posterior.
E inla 'poisson' likelihood exposure parameter
Ntrials inla 'binomial' likelihood parameter
control.compute INLA option, See control.compute
control.inla INLA option, See control.inla
control.fixed INLA option, See control.fixed
... Additional options passed on to inla

Author(s)

Fabian E. Bachl <<bachlfab@gmail.com>>
Examples

```r
# Generate default bru options
opts = bru.options()

# Print them:
opts
```

---

### cprod

Cross product of integration points

Description

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

Usage

cprod(...)

Arguments

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

Value

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

Author(s)

Fabian E. Bachl <bachlfab@gmail.com>

Examples

```r
# ipoints needs INLA
if (require("INLA", quietly = TRUE)) {
  # Create integration points in dimension 'myDim' and 'myDiscreteDim'
  ips1 = ipoints(c(0,8), name = "myDim")
  ips2 = ipoints(as.integer(c(1,2,3)), name = "myDiscreteDim")

  # Calculate the cross product
```


```r
deltaIC

ips = cprod(ips1, ips2)

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

```

deltaIC

**Summarise DIC and WAIC from lgcp objects.**

**Description**

Calculates DIC and WAIC differences and produces an ordered summary.

**Usage**

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

**Arguments**

- `...` Comma-separated objects inheriting from class `inla` and obtained from a run of `inla`, `bru` or `lgcp`
- `criterion` If 'DIC', plots DIC differences; If 'WAIC', plots WAIC differences.

**Value**

A data frame with each row containing the model name, DIC, WAIC, deltaDIC, and deltaWAIC.

**Examples**

```r

# 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 two models
fit <- bru(y ~ x, "gaussian", input.df)
fit2 <- bru(y ~ x, "Poisson", input.df)

# Compare DIC

deltaIC(fit, 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 first component.
- prediction2: A prediction of the first component.
- samplers: A SpatialPolygon object describing the area for which to compute the cumulative variance measure.
- mesh: The inla.mesh for which the prediction was performed (required for cumulative Vmeasure).

Value
Variance and correlations measures.

References

Examples
if (require("INLA", quietly = TRUE)) {
  # Load Gorilla data
data("gorillas", package = "inlabru")
  # Use RColorBrewer
library(RColorBrewer)
  # Fit a model with two components:
# 1) A spatial smoothe SPDE
# 2) A spatial covariate effect (vegetation)

pcmatern <- inla.spde2.pcmatern(gorillas$mesh,
    prior.sigma = c(0.1, 0.01),
    prior.range = c(5, 0.01))

cmp <- coordinates ~ vegetation(map = gorillas$gcov$vegetation, model = "factor") +
    spde(map = coordinates, model = pcmatern, mesh = gorillas$mesh) -
    Intercept

fit <- lgcp(cmp, gorillas$nests, samplers = gorillas$boundary)

# Predict SPDE and vegetation at the mesh vertex locations

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

# Plot component mean

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

# Plot component variance

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

# Calculate variance and correlation measure

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

# Variance contribution of the components

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

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

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

# Covariance of SPDE field and vegetation
ggplot() + gg(gorillas$mesh, color = vm$cov)

# Correlation between field and vegetation
ggplot() + gg(gorillas$mesh, color = vm$cor)

# Variance and correlation integrated over space
vm.int <- devel.cvmeasure(joint, field, veg,
                           samplers = ipoints(gorillas$boundary, gorillas$mesh),
                           mesh = gorillas$mesh)
vm.int
}

---

**generate**

*Generate samples from fitted bru and inla models*

**Description**

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

**Usage**

```r
generate(object, ...)
```

**Arguments**

- `object` a fitted model.
- `...` additional arguments affecting the samples produced.

**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 sample generators: `generate.bru`, `generate.inla`
generate.bru

Examples

```r
if (require("INLA", 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(map = x, model = "linear"), "gaussian", 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$x, smp[[1]], type = "l")
  for (k in 2:length(smp)) points(df$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
  plot(do.call(rbind, smp))
}
```

generate.bru

Sampling based on bru posteriors

Description

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

```r
generate(object, data, formula = NULL, n.samples = 100, ...)
```

Arguments

- **object**: A bru object obtained by calling `bru`.
- **data**: A data.frame or SpatialPointsDataFrame of covariates needed for sampling.
- **formula**: A formula determining which effects to sample from and how to combine them analytically.
- **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.
- **...**: additional, unused arguments.

Value

List of generated samples

See Also

- `predict.bru`
- Other sample generators: `generate.inla`, `generate`

Examples

```r
if (require("INLA", 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(map = x, model = "linear"), "gaussian", 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$x, smp[[1]], type = "l")
for (k in 2:length(smp)) points(df$x, smp[[k]], type = "l")
```
# We can also draw samples from the joint posterior

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

# ... and plot them
plot(do.call(rbind, smp))

---

**generate.inla**  
Sampling based on bru posteriors

**Description**

Takes a fitted `inla` object produced by `INLA::inla()` 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**

```r
## S3 method for class 'inla'
generate(object, ...)
```

**Arguments**

- `object`  
  An `inla` object obtained by calling `INLA::inla()`.
- `...`  
  additional arguments passed on to `generate.bru`.

**Value**

List of generated samples

**Author(s)**

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

**See Also**

- `predict.inla`
- Other sample generators: `generate.bru, generate`
Examples

```r
# Some features use the INLA package.
if (require("INLA", 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 Gaussian likelihood model
  formula = y ~ x
  fit <- inla(formula, "gaussian", data = input.df, control.compute=list(config = TRUE))

  # Generate samples from the posterior distribution of exp(x), where x is the fixed effect.
  xpost = generate(fit, NULL, ~ exp(x), n.samples = 2)
  xpost
  plot(xpost[[1]])
}
```

---

**gg**

`ggplot2` geoms for `inlabru` related objects

### Description

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

### Usage

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

### Arguments

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

### Value

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

Other geomes for inla and inlabru predictions: gg.data.frame, gg.matrix, gg.prediction, gm
Other geomes for spatial data: gg.SpatialGridDataFrame, gg.SpatialLines, gg.SpatialPixelsDataFrame, gg.SpatialPoints, gg.SpatialPolygons, gm
Other geomes for meshes: gg.inla.mesh.1d, gg.inla.mesh, gm
Other geomes for Raster data: gg.RasterLayer, gm

Examples

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

# Invoke ggplot and add geomes for the Gorilla nests and the survey boundary
ggplot() + gg(gorillas$boundary) + gg(gorillas$nests)

---

**gg.data.frame**

**Geom for data.frame**

Description

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

Usage

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

Arguments

... Arguments passed on to gg.prediction.

Value

Concatenation of a geom_line value and optionally a geom_ribbon value.

See Also

Other geomes for inla and inlabru predictions: gg.matrix, gg.prediction, gg, gm
Examples

# 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, "gaussian", input.df)

# Predict posterior statistics of 'x'

xpost = predict(fit, formula = ~ x)

# 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 simulatenneously estimate posteriors
# of multiple variables:

xipost = predict(fit, formula = ~ data.frame(post = c(Intercept, x)))

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

rownames(xipost) = c("Intercept","x")
p1 = ggplot() + gg(xipost, bar = TRUE)
p1

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

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

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

ggplot() + gg(xipost)

# For ease of use we can also type

plot(xipost)
gg.inla.mesh

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

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

# Much like the other geoms 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.inla.mesh  

\textit{Geom for inla.mesh objects}

Description

This function extracts the graph of an inla.mesh object and uses \texttt{geom_line} to visualize the graph’s edges. Alternatively, if the color argument is provided, interpolates the colors across for a set of \texttt{SpatialPixels} covering the mesh area and calls \texttt{gg.SpatialPixelDataFrame()} to plot the interpolation.

Usage

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

Arguments

data  
A \texttt{inla.mesh} object.

color  
A vector of scalar values to fill the mesh with colors. The length of the vector must correspond to the number of mesh vertices.

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

edge.color  
Color of the mesh edges.

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

int.color  
Color used to plot the interior boundaries.

exterior  
If TRUE, plot the exterior boundaries of the mesh.

ext.color  
Color used to plot the interior boundaries.

crs  
A \texttt{CRS} object defining the coordinate system to project the mesh to before plotting.
gg.inla.mesh.1d

mask A SpatialPolygon defining the region that is plotted.
nx Number of pixels in x direction (when plotting using the color parameter).
ny Number of pixels in y direction (when plotting using the color parameter).
... ignored arguments (S3 generic compatibility).

Value

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

See Also

Other geomes for meshes: gg.inla.mesh.1d, gg, gm

Examples

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

# Plot mesh using default edge colors
ggplot() + gg(gorillas$mesh)

# Don't show interior and exterior boundaries

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

# Change the edge colors

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

# Use the x-coordinate of the vertices to colorize the triangles and
# mask the plotted area by the survey boundary, i.e. only plot the inside
xcoord = gorillas$mesh$loc[,1]
ggplot() +
   gg(gorillas$mesh, color = (xcoord-580), mask = gorillas$boundary) +
   gg(gorillas$boundary)

---

gg.inla.mesh.1d Geom for inla.mesh.1d objects

Description

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

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

```r
gg(data, mapping = aes_string("x", "y"), y = 0,
    shape = 4, ...)
```

Arguments

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

Value

An object generated by `geom_point`.

See Also

Other geomes for meshes: `gg.inla.mesh`, `gg`, `gm`

Examples

```r
# Some features use the INLA package.
if (require("INLA", quietly = TRUE)) {

    # Load INLA
    library(INLA)

    # Create a 1D mesh
    mesh = inla.mesh.1d(seq(0,10,by=0.5))

    # Plot it
    ggplot() + gg(mesh)

    # Plot it using a different shape and size for the mesh nodes
    ggplot() + gg(mesh, shape = "|", size = 5)
}
```
gg.matrix

**Geom for matrix**

**Description**

Creates a tile geom for plotting a matrix.

**Usage**

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

**Arguments**

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

**Value**

A `geom_tile` with reversed y scale.

**See Also**

Other geomes for inla and inlabru predictions: `gg.data.frame`, `gg.prediction`, `gg.gm`

**Examples**

```r
A = matrix(runif(100), nrow = 10)
ggplot() + gg(A)
```

---

gg.prediction

**Geom for predictions**

**Description**

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

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

## S3 method for class 'prediction'

\[
\text{gg}(\text{data}, \text{mapping} = \text{NULL}, \text{ribbon} = \text{TRUE},
\begin{align*}
&\text{alpha} = 0.3, \text{bar} = \text{FALSE}, \ldots
\end{align*}
\]

Arguments

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

Value

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

See Also

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

Examples

```r
# 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, "gaussian", input.df)

# Predict posterior statistics of 'x'
xpost = predict(fit, formula = ~ x)

# 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:

```r
xipost = predict(fit, formula = ~ data.frame(post = c(Intercept, x)))
xipost
```

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

```r
rownames(xipost) = c("Intercept","x")
p1 = ggplot() + gg(xipost, bar = TRUE)
p1
```

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

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

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

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

# For ease of use we can also type

```
plot(xipost)
```

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

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

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

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

---

**gg.RasterLayer**

*Geom for RasterLayer objects*

**Description**

This function takes a RasterLayer object, converts it into a SpatialPixelsDataFrame and uses \code{geom_tile} to plot the data.
Usage

## S3 method for class 'RasterLayer'

`gg(data, mapping = aes_string(x = "x", y = "y",
    fill = "layer"), ...)`

Arguments

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

Details

This function requires the ‘raster’ package.

Value

An object returned by `geom_tile`

See Also

Other geomes for Raster data: `gg`, `gm`

Examples

# Some features require the raster and spatstat.data packages.
if (require("spatstat.data", quietly = TRUE) &&
    require("raster", quietly = TRUE)) {

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

    # Convert elevation covariate to RasterLayer
elev = as(gorillas.extra$elevation, "RasterLayer")

    # Plot the elevation
    ggplot() + gg(elev)

}
gg.SpatialGridDataFrame

Geom for SpatialGridDataFrame objects

Description
Coerces input SpatialGridDataFrame to SpatialPixelsDataFrame and calls gg.SpatialPixelDataFrame to plot it.

Usage
## S3 method for class 'SpatialGridDataFrame'

`gg(data, ...)`

Arguments
data  A SpatialGridDataFrame object.
...  Arguments passed on to gg.SpatialPixelsDataFrame().

Value
A `geom_tile` value.

See Also

Examples

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

# Plot Gorilla elevation covariate provided as SpatialPixelsDataFrame. # The same syntax applies to SpatialGridDataFrame objects.
ggplot() + gg(gorillas$gcov$elevation)

# Add Gorilla survey boundary and nest sightings
ggplot() +
  gg(gorillas$gcov$elevation) +
  gg(gorillas$boundary) +
  gg(gorillas$nests)

# Load pantropical dolphin data
```
data("mexdolphin")

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

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

# Change color

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

# Visualize data annotations: line width by segment number

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

ggplot() + gg(mexdolphin$samplers, aes(color = seg))

# Visualize data annotations: point size by dolphin group size

names(mexdolphin$points) # 'size' holds the group size

ggplot() + gg(mexdolphin$points, aes(size = size))

---

**gg.SpatialLines**

*Geom for SpatialLines objects*

**Description**

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

**Usage**

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

**Arguments**

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

A `geom_segment` return value.

See Also


Examples

# Load Gorilla data

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

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

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

# Add Gorilla survey boundary and nest sightings

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

# Load pantropical dolphin data

data("mexdolphin")

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

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

# Change color

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

# Visualize data annotations: line width by segment number

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

# Visualize data annotations: point size by dolphin group size
names(mexdolphin$points) # 'size' holds the group size
ggplot() + gg(mexdolphin$points, aes(size = size))

---

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

### Description

Uses `geom_point` to plot the pixel centers.

### Usage

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

### Arguments

- **data**: A `SpatialPixels` object.
- **...**: Arguments passed on to `geom_tile`.

### Value

A `geom_tile` return value.

### See Also


### Examples

```r
# Load Gorilla data
data(gorillas)

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

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

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

Usage

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

 gg(data, mapping = NULL, alpha = NULL,
      crs = NULL, mask = NULL, ...)
```

Arguments

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

Value

A geom_tile return value.

See Also


Examples

```r
# Load Gorilla data

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

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

 ggplot() + gg(gorillas$gcov$elevation)
```
```r
# Add Gorilla survey boundary and nest sightings

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

# Load pantropical dolphin data

data("mexdolphin")

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

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

# Change color

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

# Visualize data annotations: line width by segment number

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

# Visualize data annotations: point size by dolphin group size

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

---

**Description**

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

**Usage**

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

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

Arguments

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

Value

A geom_point return value

See Also


Examples

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

# Plot Gorilla elevation covariate provided as SpatialPixelsDataFrame. The same syntax applies to SpatialGridDataFrame objects.
ggplot() + gg(gorillas$gcov$elevation)

# Add Gorilla survey boundary and nest sightings

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

# Load pantropical dolphin data
data("mexdolphin")

# 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.SpatialPolygons

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

# Visualize data annotations: line width by segment number

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

# Visualize data annotations: point size by dolphin group size

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

---

**gg.SpatialPolygons**  
*Geom for SpatialPolygons objects*

**Description**

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

**Usage**

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

**Arguments**

- **data** A SpatialPolygons object.
- **mapping** Aesthetic mappings created by `aes` or `aes_` used to update the default mapping. The default mapping is `aes_string(x = "long", y = "lat", group = "group")`.
- **crs** A CRS object defining the coordinate system to project the data to before plotting.
- **color** Filling color for the polygons.
- **alpha** Alpha level for polygon filling.
- **...** Arguments passed on to `geom_polygon`.

**Value**

A `geom_polygon` return value.
gg.SpatialPolygons

See Also


Examples

# Load Gorilla data
data("gorillas", package = "inlabru")
# Plot Gorilla elevation covariate provided as SpatialPixelsDataFrame.
# The same syntax applies to SpatialGridDataFrame objects.
ggplot() + gg(gorillas$gcov$elevation)
# Add Gorilla survey boundary and nest sightings
ggplot() +
  gg(gorillas$gcov$elevation) +
  gg(gorillas$boundary) +
  gg(gorillas$nest)
# Load pantropical dolphin data
data("mexdolphin")
# Plot the pantropical survey boundary, ship transects and dolphin sightings
ggplot() +
  gg(mexdolphin$ppoly) + # survey boundary as SpatialPolygon
  gg(mexdolphin$samplers) + # ship transects as SpatialLines
  gg(mexdolphin$points) # dolphin sightings as SpatialPoints
# Change color
ggplot() +
  gg(mexdolphin$ppoly, color = "green") + # survey boundary as SpatialPolygon
  gg(mexdolphin$samplers, color = "red") + # ship transects as SpatialLines
  gg(mexdolphin$points, color = "blue") # dolphin sightings as SpatialPoints
# Visualize data annotations: line width by segment number
names(mexdolphin$samplers) # 'seg' holds the segment number
  ggplot() + gg(mexdolphin$samplers, aes(color = seg))
# Visualize data annotations: point size by dolphin group size
names(mexdolphin$points) # 'size' holds the group size
  ggplot() + gg(mexdolphin$points, aes(size = size))
globe

Plot a globe using rgl

Description

Creates a textured sphere and lon/lat coordinate annotations.

Usage

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

Arguments

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

Details

This function requires the ‘rgl’ and ‘sphereplot’ packages.

Value

No value, used for plotting side effect.

See Also

Other inlabru RGL tools: glplot.SpatialLines, glplot.SpatialPoints, glplot.inla.mesh, glplot

Examples

if (require("rgl", quietly = TRUE) &&
   require("sphereplot", quietly = TRUE)) {

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

   # Show the globe
glplot

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

glplot(mexdolphin$mesh)
glplot(mexdolphin$samplers)
glplot(mexdolphin$points)

---

**glplot**  
*Render Spatial* and *inla.mesh* objects using RGL*

**Description**

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

**Usage**

`glplot(object, ...)`

**Arguments**

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

**See Also**

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

**Examples**

```r
if (require("rgl", quietly = TRUE) && require("sphereplot", quietly = TRUE)) {
  # Load pantropoical dolphin data
data("mexdolphin", package = "inlabru")
  # Show the globe
globe()
  # Add mesh, ship transects and dolphin sightings stored
```
# as inla.mesh, SpatialLines and SpatialPoints objects, respectively

glpot(mexdolphin$mesh)
glpot(mexdolphin$samplers)
glpot(mexdolphin$points)
}

glpot.inla.mesh  

Visualize SpatialPoints using RGL

Description

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

Usage

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

Arguments

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

See Also

Other inlabru RGL tools: globe, glplot.SpatialLines, glplot.SpatialPoints, glplot

Examples

if (require("rgl", quietly = TRUE) &
require("sphereplot", quietly = TRUE)) {

# Load pantropoical dolphin data

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

# Show the globe
globe()
# Add mesh, ship transects and dolphin sightings stored 
# as inla.mesh, SpatialLines and SpatialPoints objects, respectively

glplot(mexdolphin$mesh)
glplot(mexdolphin$samplers)
glplot(mexdolphin$points)

---

### glplot.SpatialLines

**Visualize SpatialLines using RGL**

#### Description

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

#### Usage

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

#### Arguments

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

#### See Also

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

#### Examples

```r
if (require("rgl", quietly = TRUE) &
    require("sphereplot", quietly = TRUE)) {

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

  # Show the globe
globe()

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

Visualize SpatialPoints using RGL

Description

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

Usage

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

Arguments

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

See Also

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

Examples

```r
if (require("rgl", quietly = TRUE) && require("sphereplot", quietly = TRUE)) {

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

  # Show the globe
  globe()

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

  glplot(mexdolphin$mesh)
  glplot(mexdolphin$samplers)
  glplot(mexdolphin$points)
}
```
gm

**ggplot geom for spatial data**

**Description**

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

**Usage**

`gm(data, ...)`

**Arguments**

- `data` an object for which to generate a geom.
- `...` Arguments passed on to gg.

**Value**

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

**See Also**

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


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

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

**Examples**

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

# Create a base map centered around the nests and plot the boundary as well as the nests gmap(gorillas$nests, maptype = "satellite") +
```
gmap

Plot a map using extend of a spatial object

Description

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

Usage

gmap(data, ...)

Arguments

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

Details

This function requires the ‘ggmap’ package.

Value

a ggplot object

Examples

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

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

*Gorilla Nesting Sites.*

**Description**

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

**Usage**

data(gorillas)

**Format**

The data are a list that contains these elements:

- **nests**: A SpatialPointsDataFrame object containing the locations of the gorilla nests.
- **boundary**: An SpatialPolygonsDataFrame object defining the boundary of the region that was searched for the nests.
- **mesh**: An inla.mesh object containing a mesh that can be used with function lgcp to fit a LGCP to the nest data.
- **gcov**: A list of SpatialGridDataFrame objects, one for each of these spatial covariates:
  - **aspect**: Compass direction of the terrain slope. Categorical, with levels N, NE, E, SE, S, SW, W and NW, which are coded as integers 1 to 8.
  - **elevation**: Digital elevation of terrain, in metres.
  - **heat**: Heat Load Index at each point on the surface (Beer's aspect), discretised. Categorical with values Warmest (Beer's aspect between 0 and 0.999), Moderate (Beer's aspect between 1 and 1.999), Coolest (Beer's aspect equals 2). These are coded as integers 1, 2 and 3, in that order.
  - **slopangle**: Terrain slope, in degrees.
  - **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.
- **plotsample**: Plot sample of gorilla nests, sampling 9x9 over the region, with 60% coverage. Components:
  - **counts**: A SpatialPointsDataFrame frame with elements x, y, count, exposure, being the x- and y-coordinates of the centre of each plot, the count in each plot and the area of each plot.
  - **plots**: A SpatialPolygonsDataFrame defining the individual plot boundaries.
  - **nests**: A SpatialPointsDataFrame giving the locations of each detected nest.
Source
Library spatstat.

References

Examples
```r
data(gorillas, package = "inlabru") # get the data
# extract all the objects, for convenience:

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

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

# Plot the plot sample
ggplot() + gg(gorillas$plotsample$plots) + gg(gorillas$plotsample$nests)
```

init.tutorial  

Global setting for tutorial sessions

Description
Increases verbosity and sets the inference strategy to empirical Bayes.

Usage
```r
init.tutorial()
```

Author(s)
Fabian E. Bachl <<bachlfab@gmail.com>>

Examples
```r
## Not run:
# Note: Only run this if you want to change the inlabru options for this session

# Determine current bru default:
bo = bru.options()
```
# By default, INLA's integration strategy is set to the INLA default 'auto':
bo$inla.options$control.inla

# Now, let's run init.tutorial() to make empirical Bayes the default
# integration method when \code{bru} calls \code{inla}

init.tutorial()

# Check if it worked:
bru.options()$inla.options$control.inla

## End(Not run)

inlabru

---

## Description

Convenient model fitting using (iterated) INLA.

## Details

inlabru facilitates Bayesian spatial modeling using integrated nested Laplace approximations. It is heavily based on R-inla (http://www.r-inla.org) but adds additional modeling abilities. Tutorials and more information can be found at http://www.inlabru.org/.

The main function for inference using inlabru is bru. For point process inference lgcp is a good starting point. The package comes with multiple real world data sets, namely gorillas, mxdolphin, seals. Plotting these data sets is straight forward using inlabru's extensions to ggplot2, e.g. the gg function. For educational purposes some simulated data sets are available as well, e.g. Poisson1_1D, Poisson2_1D, Poisson2_1D and toygroups.

## Author(s)

Fabian E. Bachl <<bachlfab@gmail.com>>

---

int

Weighted summation (integration) of data frame subsets
Description

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

Usage

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

Arguments

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

Value

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

Examples

```r
# ipoints needs INLA
if (require("INLA", quietly = TRUE)) {
  # Create integration points in two dimensions, x and y
  ips = cprod(ipoints(c(0,10), 10, name = "x"),
                ipoints(c(1,5), 10, name = "y"))

  # The sizes of the domains are 10 and 4 for x and y, respectively.
  # Integrating f(x,y) = 1 along x and y should result in the total
  # domain size 40
  int(ips, rep(1, nrow(ips)), c("x","y"))
}
```
ipoints

Generate integration points

Description

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

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

The domain parameter is an inla.mesh.1d or inla.mesh object that can be employed to project the integration points to the vertices of the mesh. This reduces the final number of integration points and reduces the computational cost of the integration. The projection can also prevent numerical issues in spatial LGCP models where each observed point is ideally surrounded by three integration point sitting at the corresponding mesh vertices. For convenience, the domain parameter can also be a single integer setting the number of equally spaced integration points in the one-dimensional case.

Usage

ipoints(region = NULL, domain = NULL, name = "x", group = NULL, project)

Arguments

region Description of the integration region boundary. In 1D either a vector of two numerics or a two-column matrix where each row describes and interval. In 2D either a SpatialPolygon or a SpatialLinesDataFrame with a weight column defining the width of the line.

domain In 1D a single numeric setting the number of integration points or an inla.mesh.1d defining the locations to project the integration points to. In 2D domain has to be an inla.mesh object describing the projection and granularity of the integration.

name Character array stating the name of the domains dimension(s)

group Column names of the region object (if applicable) for which the integration points are calculated independently and not merged by the projection.

project If TRUE, project the integration points to mesh vertices

Value

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

Author(s)

Fabian E. Bachl <bachlfab@gmail.com>
Examples

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

# Create 50 integration points covering the dimension 'myDim' between 0 and 10.
ips = ipoints(c(0,10), 50, name = "myDim")
plot(ips)

# Create integration points for the two intervals [0,3] and [5,10]
ips = ipoints(matrix(c(0,3, 5,10), nrow = 2, byrow = TRUE), 50)
plot(ips)

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

# Obtain 2D integration points from a SpatialPolygon
data(gorillas, package = "inlabru")
ips = ipoints(gorillas$boundary)
ggplot() + gg(gorillas$boundary) + gg(ips, aes(size = weight))

' Project integration points to mesh vertices
ips = ipoints(gorillas$boundary, domain = gorillas$mesh)
ggplot() + gg(gorillas$mesh) + gg(gorillas$boundary) + gg(ips, aes(size = weight))

# Turn a 2D mesh into integration points
ips = ipoints(gorillas$mesh)
ggplot() + gg(gorillas$boundary) + gg(ips, aes(size = weight))
}
```

**lgcp**  
_Log Gaussian Cox process (LGCP) inference using INLA_

**Description**

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

Usage

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

Arguments

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

Value

An bru object

Examples

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

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

if (require("INLA", quietly = TRUE)) {
```
# Define SPDE prior
matern <- inla.spde2.pcmatern(gorillas$mesh,
    prior.sigma = c(0.1, 0.01),
    prior.range = c(5, 0.01))

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

# Fit the model
fit <- lgcp(cmp, gorillas$nests, samplers = gorillas$boundary)

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

# Plot the intensity

like

library(brugge)

like(family, formula = . ~ ., data = NULL, components = NULL,
    mesh = NULL, E = NULL, Ntrials = NULL, samplers = NULL,
    ips = NULL, domain = NULL, options = list())

Description

Likelihood construction for usage with bru

Usage

like(family, formula = . ~ ., data = NULL, components = NULL,
    mesh = NULL, E = NULL, Ntrials = NULL, samplers = NULL,
    ips = NULL, domain = NULL, options = list())

Arguments

family A character identifying a valid inla likelihood. Alternatively 'ep' for Cox processes.
formula a formula where the right hand side expression defines the predictor used in the optimization.
data Likelihood-specific data.
components Components.
mesh An inla.mesh object.
like

E

Exposure parameter for family = 'poisson' passed on to inla. Special case if family is 'cp': rescale all integration weights by E. Default taken from options$E.

Ntrials

A vector containing the number of trials for the 'binomial' likelihood. Default value is rep(1, n.data). Default taken from options$Ntrials.

samplers

Integration domain for 'cp' family.

ips

Integration points for 'cp' family. Overrides samplers.

domain

Named list of domain definitions.

options

list of global options overriding bru.options

Value

A likelihood configuration which can be used to parameterize bru.

Author(s)

Fabian E. Bachl <<bachlfab@gmail.com>>

Examples

if (require("INLA", quietly = TRUE)) {

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

# Generate the data 
set.seed(123)

n1 = 200
n2 = 10

x1 = runif(n1)
x2 = runif(n2)
z2 = runif(n2)

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

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

# Single likelihood models and inference using bru are done via 

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

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

# A joint model has two likelihoods, which are set up using the like function
lik1 = like("gaussian", formula = y ~ x + Intercept, data = df1)
lik2 = like("poisson", formula = y ~ x + z + Intercept, data = df2)

# The union of effects of both models gives the components needed to run bru
jcmp = y ~ x + z + Intercept
jfit = bru(jcmp, lik1, lik2)

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

mexdolphin

Pan-tropical spotted dolphins in the Gulf of Mexico.

Description

This a version of the mexdolphins dataset from the package dsm, reformatted as point process data for use with inlabru. The data are from a combination of several NOAA shipboard surveys conducted on pan-tropical spotted dolphins in the Gulf of Mexico. 47 observations of groups of dolphins were detected. The group size was recorded, as well as the Beaufort sea state at the time of the observation.

Format

A list of objects:

points: A SpatialPointsDataFrame object containing the locations of detected dolphin groups, with their size as an attribute.
samplers: A SpatialLinesDataFrame object containing the transect lines that were surveyed.
mesh: An inla.mesh object containing a Delaunay triangulation mesh (a type of discretization of continuous space) covering the survey region.
ppoly: An SpatialPolygonsDataFrame object defining the boundary of the survey region.
simulated: A SpatialPointsDataFrame object containing the locations of a simulated population of dolphin groups. The population was simulated from a 'codeinlabru model fitted to the actual survey data. Note that the simulated data do not have any associated size information.
Source

Library dsm.

References


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

Examples

```r
data(mexdolphin, package="inlabru")
plot(mexdolphin$mesh, edge.color="lightgray", draw.segments=FALSE) # draw mesh
plot(mexdolphin$ppoly, add=TRUE) # add survey region boundary
plot(mexdolphin$samplers, col="blue", add=TRUE) # draw transects (in and out of survey region)
grsize = attributes(mexdolphin$points)$data[, "size"] # Get group size data
plot(mexdolphin$points, pch=19, col="red", cex=log(grsize/30), add=TRUE)
```

mrsea

Marine renewables strategic environmental assessment

Description

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

Format

A list of objects:

points: A SpatialPointsDataFrame object containing the locations of XXXXX.
samplers: A SpatialLinesDataFrame object containing the transect lines that were surveyed.
mesh: An inla.mesh object containing a Delaunay triangulation mesh (a type of discretization of continuous space) covering the survey region.
boundary: An SpatialPolygonsDataFrame object defining the boundary of the survey region.
covar: An SpatialPointsDataFrame containing sea depth estimates.

Source

Library MRSea.
**multiplot**

**References**

NONE YET

**Examples**

```r
data(mrsea)
ggplot() + gg(mrsea$mesh) + gg(mrsea$samplers) + gg(mrsea$points) + gg(mrsea$boundary)
```

---

**multiplot**  
*Multiple ggplots on a page.*

**Description**

Renders multiple ggplots on a single page.

**Usage**

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

**Arguments**

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

**Author(s)**

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

**Source**

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

**Examples**

```r
df = data.frame(x=1:10, y=1:10, z=11:20)
pl1 = ggplot(data = df) + geom_line(mapping = aes(x,y), color = "red")
pl2 = ggplot(data = df) + geom_line(mapping = aes(x,z), color = "blue")
multiplot(pl1,pl2, cols = 2)
```
pixels Generate SpatialPixels covering an inla.mesh

Description
Generate SpatialPixels covering an inla.mesh

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

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

Value
SpatialPixels covering the mesh

Author(s)
Fabian E. Bachl <<bachlfab@gmail.com>>

Examples

data("mrsea")
pxl = pixels(mrsea$mesh, nx = 50, ny = 50)
ggplot() + gg(pxl) + gg(mrsea$mesh)

plot.bru Plot method for posterior marginals estimated by bru

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

Usage

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

Arguments

- **x**: a fitted bru model.
- **...**: A character naming the effect to plot, e.g. "Intercept".

Value

an object of class gg

Examples

## Not run:

# Generate some data and fit a simple model
input.df <- data.frame(x=cos(1:10))
input.df <- within(input.df, y <- 5 + 2*cos(1:10) + rnorm(10, mean=0, sd=0.1))
fit <- bru(y ~ x, "gaussian", input.df)
summary(fit)

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

## End(Not run)

---

plot.prediction

Plot prediction using ggplot2

Description

Generates a base ggplot2 using ggplot and adds a geom for input x using gg.

Usage

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

Arguments

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

an object of class gg

**Examples**

```r
# 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, "gaussian", input.df)

# Predict posterior statistics of 'x'
xpost = predict(fit, formula = ~ x)

# 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 simulatenneously estimate posteriors
# of multiple variables:
xipost = predict(fit, formula = ~ data.frame(post = c(Intercept, x)))
xipost

# If we still want a plot in the previous style we have to set the bar parameter to TRUE
rownames(xipost) = c("Intercept","x")
p1 = ggplot() + gg(xipost, bar = TRUE)
p1

# Note that gg also understands the posterior estimates generated while running INLA
p2 = ggplot() + gg(fit$summary.fixed, bar = TRUE)
multiplot(p1, p2)

# By default, if the prediction has more than one row, gg will plot the column 'mean' against
# the row index. This is for instance usefuul for predicting and plotting function
# but not very meaningful given the above example:
ggplot() + gg(xipost)
```
# For ease of use we can also type

plot(xipost)

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

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

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

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

---

**plotsample**

*Create a plot sample.*

**Description**

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

**Usage**

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

**Arguments**

- **spdf**
  - A `SpatialPointsDataFrame` defining the points that are to be sampled by the plot sample.

- **boundary**
  - A `SpatialPolygonsDataFrame` defining the survey boundary within which the points occur.

- **x.ppn**
  - The proportion of the x-axis that is to be included in the plots.

- **y.ppn**
  - The proportion of the y-axis that is to be included in the plots.

- **nx**
  - The number of plots in the x-dimension.

- **ny**
  - The number of plots in the y-dimension.
Value

A list with three components:

plots: A SpatialPolygonsDataFrame object containing the plots that were sampled.
dets: A SpatialPointsDataFrame object containing the locations of the points within the plots.
counts: A dataframe containing the following columns
  x: The x-coordinates of the centres of the plots within the boundary.
y: The y-coordinates of the centres of the plots within the boundary.
n: The numbers of points in each plot.
area: The areas of the plots within the boundary

Examples

```r
# Some features require the raster package
if (require("raster", quietly = TRUE)) {

data(gorillas, package = "inlabru")
plotpts = plotsample(gorillas$nests, gorillas$boundary, x.ppn=0.4, y.ppn=0.4, nx=5, ny=5)
ggplot() + gg(plotpts$plots) + gg(plotpts$dets, pch="+", cex=2) + gg(gorillas$boundary)
}
```

point2count

Convert a plot sample of points into one of counts.

Description

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

Usage

point2count(plots, dets)

Arguments

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

Value

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

```r
# Some features require the raster package
if (require("raster", quietly = TRUE)) {
  data(gorillas, package = "inlabru")
  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$dets)
  x=coordinates(countdata)[,1]
  y=coordinates(countdata)[,2]
  count=countdata@data$n
  p2 = ggplot() + gg(gorillas$boundary) + gg(plotpts$plots) + geom_text(aes(label=count, x=x, y=y))
  multiplot(p1,p2,cols=2)
}
```

---

**Poisson1_1D**

1-Dimensional Homogeneous Poisson example.

### Description

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

### Usage

```r
data(Poisson1_1D)
```

### Format

The data contain the following R objects:

- **lambda1_1D**: A function defining the intensity function of a nonhomogeneous Poisson process. Note that this function is only defined on the interval (0,55).
- **E_nc1**: The expected counts of the gridded data.
- **pts1**: The locations of the observed points (a data frame with one column, named `x`).
- **countdata1**: A data frame with three columns, containing the count data:
  - `x`: The grid cell midpoint.
  - `count`: The number of detections in the cell.
  - `exposure`: The width of the cell.
Examples

```r
library(ggplot2)
data(Poisson1_1D)
ggplot(countdata1) + geom_point(data = countdata1, aes(x=x, y=count), col="blue") + ylim(0, max(countdata1$count)) + geom_point(data = pts1, aes(x=x), y = 0.2, shape="|", cex=4) + geom_point(data = countdata1, aes(x=x), y = 0, shape="+", col="blue", cex=4) + xlab(expression(bold(s))) + ylab("count")
```

Poisson2_1D

1-Dimensional NonHomogeneous Poisson example.

Description

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

Usage

```r
data(Poisson2_1D)
```

Format

The data contain the following R objects:

- `lambda2_1D`: A function defining the intensity function of a nonhomogeneous Poisson process. Note that this function is only defined on the interval (0,55).
- `cov2_1D`: A function that gives what we will call a 'habitat suitability' covariate in 1D space.
- `E_nc2`: The expected counts of the gridded data.
- `pts2`: The locations of the observed points (a data frame with one column, named `x`).
- `countdata2`: A data frame with three columns, containing the count data:
  - `x`: The grid cell midpoint.
  - `count`: The number of detections in the cell.
  - `exposure`: The width of the cell.

Examples

```r
library(ggplot2)
data(Poisson2_1D)
pl = 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")
```
\begin{verbatim}
ss = seq(0,55,length=200)
lambda = lambda2_1D(ss)
p2 = ggplot() +
  geom_line(data=data.frame(x=ss,y=lambda), aes(x=x,y=y),col="blue") +
  geom_point(data = pts2, aes(x=x), y = 0.2, shape = "|",cex=4) +
  xlab(expression(bold(s))) +
  ylab(expression(lambda(bold(s))))
multiplot(p1,p2,cols=1)
\end{verbatim}

**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 grid-ded 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:
  - `x`: The grid cell midpoint.
  - `count`: The number of detections in the cell.
  - `exposure`: The width of the cell.
- `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**

```
library(ggplot2)
data(Poisson3_1D)
# first the plots for the 10-bin case:
pl1 = 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) +
```
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.

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

Usage

## S3 method for class 'bru'
predict(object, data = NULL, formula = NULL,
         n.samples = 100, ...)

Arguments

object An object obtained by calling bru or lgcp.
data A data.frame or SpatialPointsDataFrame of covariates needed for the prediction.
formula  A formula determining which effects to predict and how to combine them.
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.
...  ignored arguments (S3 generic compatibility).

Details
Internally, this method calls \code{generate.bru} in order to draw samples from the model.

Value
a \code{data.frame} or Spatial* object with predicted mean values and other summary statistics attached.

Examples

if (require("INLA", quietly = TRUE)) {
  # Load the Gorilla data
  data(gorillas, package ="inlabru")
  # Plot the Gorilla nests, the mesh and the survey boundary
  ggplot() +
    gg(gorillas$mesh) +
    gg(gorillas$nests) +
    gg(gorillas$boundary) +
    coord_fixed()
  # Define SPDE prior
  matern <- inla.spde2.pcmatern(gorillas$mesh,
                               prior.sigma = c(0.1, 0.01),
                               prior.range = c(5, 0.01))
  # Define domain of the LGCP as well as the model components (spatial SPDE effect and Intercept)
  cmp <- coordinates ~ mySmooth(map = coordinates, model = matern) + Intercept
  # Fit the model
  fit <- lgcp(cmp, gorillas$nests, samplers = gorillas$boundary)
  # 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, ~ Intercept)
  rownames(icpt) = "Intercept"
  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, ~ exp(Intercept))
rownames(exp.icpt) = "exp(Intercept)"
plot(rbind(icpt, exp.icpt), bar = TRUE)

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

t = vertices(gorillas$mesh)

# we obtain these vertices as a \code{SpatialPointsDataFrame}

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

# Predicting 'mySmooth' at these locations works as follows

mySmooth = predict(fit, t, ~ mySmooth)

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

class(mySmooth)
head(t)
head(mySmooth)

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

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

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

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

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

t = pixels(gorillas$mesh)
mySmooth = predict(fit, t, ~ mySmooth)

# This will give us a \code{SpatialPixelDataFrame} with the columns we are looking for

head(mySmooth)
ggplot() + gg(mySmooth)
**predict.inla**  
*Prediction from fitted inla model*

**Description**
Takes a fitted inla object produced by the function inla() 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 posterior of the estimated effects.

**Usage**
```r
## S3 method for class 'inla'
predict(object, ...)
```

**Arguments**
- `object`  
  A bru object obtained by calling bru or lgcp.
- `...`  
  Arguments passed on to predict.bru.

**Value**
A prediction object.

**Author(s)**
Fabian E. Bachl <bachlfab@gmail.com>

**Examples**
```r
# Some features use the INLA package.
if (require("INLA", 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 Gaussian likelihood model
  formula = y ~ x
  fit <- inla(formula, "gaussian", data = input.df, control.compute=list(config = TRUE))
  # Estimate posterior statistics of exp(x), where x is the fixed effect.
  xpost = predict(fit, NULL, ~ exp(x))
```
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 \texttt{inla.mesh.1d} or \texttt{inla.mesh} object. In between mesh nodes the log intensity is assumed to be linear.

**Usage**

\begin{verbatim}
sample.lgcp(mesh, loglambda, strategy = NULL, R = NULL,
             samplers = NULL, ignore.CRS = FALSE)
\end{verbatim}

**Arguments**

- \texttt{mesh} \hspace{1cm} An \texttt{inla.mesh} object
- \texttt{loglambda} \hspace{1cm} vector or matrix; A vector of log intensities at the mesh vertices (for higher order basis functions, e.g. for \texttt{inla.mesh.1d} meshes, \texttt{loglambda} should be given as \texttt{mesh$m} basis function weights rather than the values at the \texttt{mesh$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.
- \texttt{strategy} \hspace{1cm} 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 \texttt{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.
- \texttt{R} \hspace{1cm} Numerical value only applicable to spherical and geographical meshes. It is interpreted as \texttt{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.
- \texttt{samplers} \hspace{1cm} A 'SpatialPolygonsDataFrame' or 'inla.mesh' object. Simulated points that fall outside these polygons are discarded.
- \texttt{ignore.CRS} \hspace{1cm} logical; if \texttt{TRUE}, ignore any CRS information in the mesh. Default \texttt{FALSE}. This affects \texttt{R} and the permitted values for \texttt{strategy}. 

---

**Description**

Sample from an inhomogeneous Poisson process
Details

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

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

Value

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

Author(s)

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

Examples

```r
# The INLA package is required
if (require("INLA", quietly = TRUE)) {

  vertices = seq(0, 3, by = 0.1)
  mesh = inla.mesh.1d(vertices)
  loglambda = 5-0.5*vertices
  pts = sample.lgcp(mesh, loglambda)
  pts$y = 0
  plot(vertices, exp(loglambda), type = "l", ylim = c(0,150))
  points(pts, pch = "|")
}

# The INLA package is required
if (require("INLA", quietly = TRUE)) {

data("gorillas", package = "inlabru")
pts = sample.lgcp(gorillas$mesh,
                 loglambda = 1.5,
```
Description

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

Usage

data(seals)

Format

The data contain these objects:

points: A SpatialPointsDataFrame Center locations of the photos
mesh: An inla.mesh enclosing the plane’s transect
ice.data: An SpatialPointsDataFrame with MODIS ice concentration estimates
ice.cv: An covdata object with interpolated ice coverage data

Source

Martin Jullum <<Martin.Jullum@nr.no>>

References


Examples

data(seals)
ggplot() + gg(seals$mesh) + gg(seals$points)
Description

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

Usage

\[
sline(data, \text{start.cols}, \text{end.cols}, \text{crs} = \text{CRS(as.character(NA))}, \\
\text{to.crs} = \text{NULL})
\]

Arguments

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

Value

SpatialLinesDataFrame

Examples

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

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

# Plot the lines
spatial.to.ppp

Convert SpatialPoints and boundary polygon to spatstat ppp object

Description

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

Usage

spatial.to.ppp(points, samplers)

Arguments

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

Value

A spatstat spatstat ppp object

Examples

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

# Use nest locations and survey boundary to create a spatstat ppp object
gp <- spatial.to.ppp(gorillas$nests, gorillas$boundary)
class(gp)

# Plot it
plot(gp)
spde.posterior

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")

Arguments

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

Value

A prediction object.

Author(s)

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

Examples

if (require("INLA", 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 an SPDE component
x <- seq(0, 55, length = 20)
mesh1D <- inla.mesh.1d(x, boundary = "free")
mdl <- x ~ spde1D(map = x, model = inla.spde2.matern(mesh1D)) + Intercept
fit <- lgcp(mdl, pts2, domain = list(x = c(0,55)))

# Calculate and plot the posterior range
range = spde.posterior(fit, "spde1D", "range")
plot(range)

# Calculate and plot the posterior log range
lrange = spde.posterior(fit, "spde1D", "log.range")
plot(lrange)

# Calculate and plot the posterior variance
variance = spde.posterior(fit, "spde1D", "variance")
plot(variance)

# Calculate and plot the posterior log variance
lvariance = spde.posterior(fit, "spde1D", "log.variance")
plot(lvariance)

# Calculate and plot the posterior Matern correlation
matcor = spde.posterior(fit, "spde1D", "matern.correlation")
plot(matcor)

# Calculate and plot the posterior Matern covariance
matcov = spde.posterior(fit, "spde1D", "matern.covariance")
plot(matcov)

spoly

---

**Description**

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

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

**Arguments**

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

**Value**

SpatialPolygonsDataFrame

**Examples**

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

# Convert to SpatialPolygonsDataFrame
pol = spoly(pts, crs = CRS(as.character(NA)))

# Plot it!
ggplot() + gg(pol)
```

---

**stransform**

Coordinate transformation for spatial objects

**Description**

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

**Usage**

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

**Arguments**

- `splist`: list of Spatial* objects
- `crs`: Coordinate reference system to change to
summary.bru

Summary for a bru fit

Description

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

Usage

## S3 method for class ‘bru’
summary(object, ...)

Arguments

object  An object obtained from a bru or lgcp call
...
  ignored arguments

Examples

if (require("INLA", quietly = TRUE)) {

  # 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, "gaussian", input.df)

  # Obtain summary
  fit$summary.fixed

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

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

  # Compare original and transformed mesh
  multiplot(ggplot() + gg(gorillas$mesh) + ggtitle("Original mesh"),
            ggplot() + gg(tmesh) + ggtitle("Transformed mesh"))

if (require("INLA", quietly = TRUE)) {
  # Alternatively, we can use the like() function to construct the likelihood:
  lik = like(family = "gaussian", data = input.df)
  fit <- bru(y ~ x + Intercept, lik)
  fit$summary.fixed
}

# An important addition to the INLA methodology is bru's ability to use non-linear predictors. Such a predictor can be formulated via like()'s formula parameter. For instance
if (require("INLA", quietly = TRUE)) {
  z = 2
  input.df <- within(input.df, y <- 5 + exp(z)*x + rnorm(10, mean=0, sd=0.1))
  lik = like(family = "gaussian", data = input.df, formula = y ~ exp(z)*x + Intercept, E = 10000)
  fit <- bru(~ z + Intercept, lik)
  # Check the result (z posterior should be around 2)
  fit$summary.fixed
}

---

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

# Load the data
data("toygroups")

# The data set is a simulation of animal groups residing in a 1D space. Their
# locations in x-space are sampled from a Cox process with intensity
ggplot(toygroups$df.intensity) + geom_line(aes(x=x, y=g.lambda))

# Adding the simulated group locations to this plot we obtain

ggplot(toygroups$df.intensity) +
  geom_line(aes(x=x, y=g.lambda)) +
  geom_point(data = toygroups$groups, aes(x, y=0), pch="|")

# Each group has a size mark attached to it.
# These group sizes are sampled from an exponential distribution
# for which the rate parameter depends on the x-coordinate

ggplot(toygroups$groups) +
  geom_point(aes(x=x, y=size))

ggplot(toygroups$df.rate) +
  geom_line(aes(x, rate))

vertices

Description

This is a generic function. The outcome depends on the object provided

Usage

vertices(object)

## S4 method for signature 'inla.mesh'
vertices(object)

Arguments

object An object for which to call the particular vertices method.

Value

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

Extract vertex locations from an inla.mesh

Description

Converts the vertices of an inla.mesh object into a SpatialPointsDataFrame.

Usage

vertices.inla.mesh(object)

Arguments

object An inla.mesh object.

Value

A SpatialPointsDataFrame of mesh vertex locations. The vrt column indicates the internal vertex id.

Author(s)

Fabian E. Bachl <bachlfab@gmail.com>

Examples

data("mrsea")
vrt = vertices(mrsea$mesh)
ggplot() + gg(mrsea$mesh) + gg(vrt, color = "red")
Index

aes, 23–25, 27, 29, 32, 34, 35
aes_, 23–25, 27, 29, 32, 34, 35
bincount, 3
bru, 3, 4, 6, 7, 9, 11, 15, 16, 46, 50–52, 56, 57, 64, 67, 76
bru.components, 5, 6
bru.options, 5, 9, 50, 52
class, 18
control.compute, 9
control.fixed, 9
control.inla, 9
countdata1 (Poisson1_1D), 61
countdata2 (Poisson2_1D), 62
countdata3a (Poisson3_1D), 63
countdata3b (Poisson3_1D), 63
cov2_1D (Poisson2_1D), 62
cprod, 10
CRS, 21, 29, 32, 34, 35
deltaIC, 11
devel.cvmeasure, 12
E_nc1 (Poisson1_1D), 61
E_nc2 (Poisson2_1D), 62
E_nc3a (Poisson3_1D), 63
E_nc3b (Poisson3_1D), 63
f, 7
formula, 5, 51
generate, 14, 16, 17
generate.bru, 14, 15, 17, 65
generate.inla, 14, 16, 17
gem_line, 19, 21, 22, 24, 25
gem_point, 22, 23, 31, 33, 34
gem_polygon, 35
gem_ribbon, 19, 24, 25
gem_segment, 29, 30
gem_tile, 24, 26–28, 31, 32

gg, 18, 19, 22–25, 27, 28, 30–32, 34, 36, 42, 46, 57
gg.data.frame, 19, 19, 24, 25, 42
gg.inla.mesh, 19, 21, 23, 42
gg.inla.mesh.1d, 19, 22, 24, 42
gg.matrix, 19, 24, 25, 42
gg.prediction, 19, 24, 24, 42, 57
gg.RasterLayer, 19, 26, 42
gg.SpatialGridDataFrame, 19, 28, 30–32, 34, 36, 42
gg.SpatialLines, 19, 28, 29, 31, 32, 34, 36, 42
gg.SpatialPixels, 19, 28, 30, 31, 32, 34, 36, 42
gg.SpatialPixelsDataFrame, 19, 28, 30, 31, 32, 34, 35, 42

geom, 57
globe, 37, 38–41
glplot, 37, 38, 39–41
glplot.inla.mesh, 37, 38, 39, 40, 41
glplot.SpatialLines, 37–39, 40, 41
glplot.SpatialPoints, 37–40, 41
gm, 19, 22–25, 27, 28, 30–32, 34, 36, 42
gmap, 42, 43
gorillas, 44, 46
init.tutorial, 45
inla, 4–6, 9, 11, 51, 52, 56
inla.mesh, 21, 68, 75
inlabru, 46
int, 46
ipoints, 10, 48
lambda1_1D (Poisson1_1D), 61
lambda2_1D (Poisson2_1D), 62
lambda3_1D (Poisson3_1D), 63
lgcp, 3, 5, 9, 11, 46, 49, 64, 67, 76

80
like, 5, 51
mexdolphin, 46, 53
mrsea, 54
multiplot, 55

pixels, 56
plot.bru, 56
plot.prediction, 57
plotsample, 59
point2count, 60
Poisson1_1D, 46, 61
Poisson2_1D, 46, 62
Poisson3_1D, 63
predict, 3, 4
predict.bru, 7, 16, 24, 25, 64, 67
predict.inla, 17, 67
pts1 (Poisson1_1D), 61
pts2 (Poisson2_1D), 62
pts3 (Poisson3_1D), 63

sample.lgcp, 68
seals, 46, 70
sline, 71
spatial.to.ppp, 72
SpatialPixels, 31
spde.posterior, 73
spoly, 74
spTransform, 75
stransform, 75
summary.bru, 76

toygroups, 46, 77

vertices, 78
vertices.inla.mesh-method (vertices), 78
vertices.inla.mesh, 79