Package 'sspm'

May 12, 2022

Type Package

Title Spatial Surplus Production Model Framework for Northern Shrimp Populations

Version 0.9.1

Description Implement a gam-based spatial surplus production model, aimed at modeling northern shrimp population in Atlantic Canada but potentially to any stock in any location. The package is opinionated in its implementation of SPMs as it internally makes the choice to use penalized spatial gams with time lags. However, it also aims to provide options for the user to customize their model.

License MIT + file LICENSE

BugReports `https://github.com/pedersen-fisheries-lab/sspm/issues`

URL `https://pedersen-fisheries-lab.github.io/sspm/`

Encoding UTF-8

RoxygenNote 7.1.2

Depends sf, mgcv, R (>= 3.5.0)

Imports stats, methods, units, checkmate, cli, tibble, magrittr, tidyr, dplyr, purrr, stringr, rlang

Suggests testthat (>= 3.0.0), covr, ggplot2, ggforce, lwgeom, tweedie, sfdct, knitr, rmarkdown

Config/testthat/edition 3

LazyData True

VignetteBuilder knitr

NeedsCompilation no

Author Valentin Lucet [aut, cre, cph],
Eric Pedersen [aut]

Maintainer Valentin Lucet <`valentin.lucet@gmail.com`>

Repository CRAN

Date/Publication 2022-05-12 08:10:02 UTC

R topics documented:

`as_discretization_method`

Cast into a `discretization_method` *object*

Description

Cast a character value into `discretization_method` object, using the list of possible methods in `spm_methods`.

Usage

```
as_discretization_method(name, method)

## S4 method for signature 'character,ANY'
as_discretization_method(name)

## S4 method for signature 'missing,`function`'
as_discretization_method(method)
```

Arguments

`name`	**[character]** The name of the method.
`method`	**[character]** If custom method, the function to use. See `spm_discretize` for more details.

Value

An objectof class `discretization_method`.

See Also

spm_methods.

Examples

```
as_discretization_method("tesselate_voronoi")
```

`borealis_simulated` *Simulated biomass data*

Description

Simulated biomass data for test and practice.

Usage

```
borealis_simulated
```

Format

A data frame:

year_f Year as a factor
sfa SFA ID number
weight_per_km2 Simualated biomass in kg per km2
temp_at_bottom Simulated water temperature
lon_dec Longitude
lat_dec Latitude
row Row ID
uniqueID Unique ID for simulated observation

`catch_simulated` *Simulated catch data*

Description

Simulated catch data for test and practice.

Usage

```
catch_simulated
```

Format

A data frame:

year_f Year as a factor
sfa SFA ID number
catch Simualated catch in kg
lon_dec Longitude
lat_dec Latitude
row Row ID
uniqueID Unique ID for simulated observation

`discretization_method-class` *sspm discretization method class*

Description

This class encapsulates a name and a method (function) used for discretization.

Slots

`name` **[character]** Name of the discretization method.

`method` **[function]** Function used for discretization.

`fit_smooths` *Fit the GAM part of a sspm model*

Description

Once formulas have been mapped onto a sspm discrete object, the GAMs can be fitted with this function. Arguments can be passed onto `bam`.

Usage

```
fit_smooths(
  sspm_object,
  boundaries,
  keep_fit = TRUE,
  predict = TRUE,
  family = mgcv::tw,
  drop.unused.levels = F,
  method = "fREML",
  ...
)

fit_spm(
  sspm_object,
  sspm_formula,
  keep_fit = TRUE,
  family = mgcv::scat,
  drop.unused.levels = F,
  select = TRUE,
  method = "REML",
  ...
)
```

```
## S4 method for signature 'sspm_dataset,sspm_discrete_boundary'
fit_smooths(
  sspm_object,
  boundaries,
  keep_fit = TRUE,
  predict = TRUE,
  family = mgcv::tw,
  drop.unused.levels = F,
  method = "fREML",
  ...
)

## S4 method for signature 'sspm,sspm_formula'
fit_spm(
  sspm_object,
  sspm_formula,
  keep_fit = TRUE,
  family = mgcv::scat,
  drop.unused.levels = F,
  select = TRUE,
  method = "REML",
  ...
)
```

Arguments

`sspm_object`	**[sspm_dataset]** An object of class sspm_dataset.
`boundaries`	**[sspm_boundary]** An object of class sspm_discrete_boundary.
`keep_fit`	**[logical]** Whether or not to keep the fitted values and model (default to TRUE, set to FALSE to reduce memory footprint).
`predict`	**[logical]** Whether or not to generate the smoothed predictions (necessary to fit the final SPM model, default to TRUE).
`family`	This is a family object specifying the distribution and link to use in fitting etc. See `glm` and `family` for more details. The extended families listed in `family.mgcv` can also be used.
`drop.unused.levels`	by default unused levels are dropped from factors before fitting. For some smooths involving factor variables you might want to turn this off. Only do so if you know what you are doing.
`method`	The smoothing parameter estimation method. `"GCV.Cp"` to use GCV for unknown scale parameter and Mallows' Cp/UBRE/AIC for known scale. `"GACV.Cp"` is equivalent, but using GACV in place of GCV. `"REML"` for REML estimation, including of unknown scale, `"P-REML"` for REML estimation, but using a Pearson estimate of the scale. `"ML"` and `"P-ML"` are similar, but using maximum likelihood in place of REML. Default `"fREML"` uses fast REML computation.
`...`	Arguments passed on to `mgcv::bam`

`formula` A GAM formula (see `formula.gam` and also `gam.models`). This is exactly like the formula for a GLM except that smooth terms, `s` and `te` can be added to the right hand side to specify that the linear predictor depends on smooth functions of predictors (or linear functionals of these).

`data` A data frame or list containing the model response variable and covariates required by the formula. By default the variables are taken from `environment(formula)`: typically the environment from which `gam` is called.

`weights` prior weights on the contribution of the data to the log likelihood. Note that a weight of 2, for example, is equivalent to having made exactly the same observation twice. If you want to reweight the contributions of each datum without changing the overall magnitude of the log likelihood, then you should normalize the weights (e.g. `weights <- weights/mean(weights)`).

`subset` an optional vector specifying a subset of observations to be used in the fitting process.

`na.action` a function which indicates what should happen when the data contain 'NA's. The default is set by the 'na.action' setting of 'options', and is 'na.fail' if that is unset. The "factory-fresh" default is 'na.omit'.

`offset` Can be used to supply a model offset for use in fitting. Note that this offset will always be completely ignored when predicting, unlike an offset included in `formula` (this used to conform to the behaviour of `lm` and `glm`).

`control` A list of fit control parameters to replace defaults returned by `gam.control`. Any control parameters not supplied stay at their default values.

`scale` If this is positive then it is taken as the known scale parameter. Negative signals that the scale paraemter is unknown. 0 signals that the scale parameter is 1 for Poisson and binomial and unknown otherwise. Note that (RE)ML methods can only work with scale parameter 1 for the Poisson and binomial cases.

`gamma` Increase above 1 to force smoother fits. `gamma` is used to multiply the effective degrees of freedom in the GCV/UBRE/AIC score (so `log(n)/2` is BIC like). `n/gamma` can be viewed as an effective sample size, which allows it to play a similar role for RE/ML smoothing parameter estimation.

`knots` this is an optional list containing user specified knot values to be used for basis construction. For most bases the user simply supplies the knots to be used, which must match up with the `k` value supplied (note that the number of knots is not always just `k`). See `tprs` for what happens in the `"tp"/"ts"` case. Different terms can use different numbers of knots, unless they share a covariate.

`sp` A vector of smoothing parameters can be provided here. Smoothing parameters must be supplied in the order that the smooth terms appear in the model formula. Negative elements indicate that the parameter should be estimated, and hence a mixture of fixed and estimated parameters is possible. If smooths share smoothing parameters then `length(sp)` must correspond to the number of underlying smoothing parameters.

`min.sp` Lower bounds can be supplied for the smoothing parameters. Note that if this option is used then the smoothing parameters `full.sp`, in the returned object, will need to be added to what is supplied here to get the smoothing parameters actually multiplying the penalties. `length(min.sp)` should always be the same as the total number of penalties (so it may be longer than `sp`, if smooths share smoothing parameters).

`paraPen` optional list specifying any penalties to be applied to parametric model terms. `gam.models` explains more.

`chunk.size` The model matrix is created in chunks of this size, rather than ever being formed whole. Reset to `4*p` if `chunk.size` < `4*p` where `p` is the number of coefficients.

`rho` An AR1 error model can be used for the residuals (based on dataframe order), of Gaussian-identity link models. This is the AR1 correlation parameter. Standardized residuals (approximately uncorrelated under correct model) returned in `std.rsd` if non zero. Also usable with other models when `discrete=TRUE`, in which case the AR model is applied to the working residuals and corresponds to a GEE approximation.

`AR.start` logical variable of same length as data, `TRUE` at first observation of an independent section of AR1 correlation. Very first observation in data frame does not need this. If `NULL` then there are no breaks in AR1 correlaion.

`discrete` with `method="fREML"` it is possible to discretize covariates for storage and efficiency reasons. If `discrete` is `TRUE`, a number or a vector of numbers for each smoother term, then discretization happens. If numbers are supplied they give the number of discretization bins.

`cluster` `bam` can compute the computationally dominant QR decomposition in parallel using parLapply from the `parallel` package, if it is supplied with a cluster on which to do this (a cluster here can be some cores of a single machine). See details and example code.

`nthreads` Number of threads to use for non-cluster computation (e.g. combining results from cluster nodes). If `NA` set to `max(1,length(cluster))`. See details.

`gc.level` to keep the memory footprint down, it can help to call the garbage collector often, but this takes a substatial amount of time. Setting this to zero means that garbage collection only happens when R decides it should. Setting to 2 gives frequent garbage collection. 1 is in between. Not as much of a problem as it used to be.

`use.chol` By default `bam` uses a very stable QR update approach to obtaining the QR decomposition of the model matrix. For well conditioned models an alternative accumulates the crossproduct of the model matrix and then finds its Choleski decomposition, at the end. This is somewhat more efficient, computationally.

`samfrac` For very large sample size Generalized additive models the number of iterations needed for the model fit can be reduced by first fitting a model to a random sample of the data, and using the results to supply starting values. This initial fit is run with sloppy convergence tolerances, so is typically very low cost. `samfrac` is the sampling fraction to use. 0.1 is often reasonable.

`coef` initial values for model coefficients

`G` if not `NULL` then this should be the object returned by a previous call to `bam` with `fit=FALSE`. Causes all other arguments to be ignored except `sp`, `chunk.size`, `gamma`,`nthreads`, `cluster`, `rho`, `gc.level`, `samfrac`, `use.chol`, `method` and `scale` (if >0).

	`fit` if `FALSE` then the model is set up for fitting but not estimated, and an object is returned, suitable for passing as the `G` argument to `bam`. `drop.intercept` Set to `TRUE` to force the model to really not have the a constant in the parametric model part, even with factor variables present.
`sspm_formula`	**[sspm_formula]** The formula specifying the the smoothing model.
`select`	Should selection penalties be added to the smooth effects, so that they can in principle be penalized out of the model? See `gamma` to increase penalization. Has the side effect that smooths no longer have a fixed effect component (improper prior from a Bayesian perspective) allowing REML comparison of models with the same fixed effect structure.

Value

A object of the class `sspm_fit`.

`map_formula` *Map model formula onto a sspm_dataset object*

Description

This functions is now used internally to map a formula onto a `sspm_dataset` or `sspm` object.

Usage

```
map_formula(data_frame, boundaries, formula, time, ...)

## S4 method for signature 'sf,ANY,formula'
map_formula(data_frame, boundaries, formula, time, ...)

## S4 method for signature 'ANY,missing,ANY'
spm_smooth(
  sspm_object,
  formula,
  boundaries,
  keep_fit = TRUE,
  predict = TRUE,
  ...
)

## S4 method for signature 'ANY,ANY,missing'
spm_smooth(
  sspm_object,
  formula,
  boundaries,
  keep_fit = TRUE,
  predict = TRUE,
```

```
  ...
)

## S4 method for signature 'ANY,ANY,sspm_boundary'
spm_smooth(
  sspm_object,
  formula,
  boundaries,
  keep_fit = TRUE,
  predict = TRUE,
  ...
)
```

Arguments

`data_frame`	**[sf data.frame]** The data.
`boundaries`	**[sspm_boundary]** An object of class sspm_discrete_boundary.
`formula`	**[formula]** A formula definition of the form response ~ smoothing_terms + ...
`time`	**[character]** The time column.
`...`	a list of variables that are the covariates that this smooth is a function of. Transformations whose form depends on the values of the data are best avoided here: e.g. `s(log(x))` is fine, but `s(I(x/sd(x)))` is not (see `predict.gam`).
`sspm_object`	**[sspm_dataset]** An object of class sspm_dataset.
`keep_fit`	**[logical]** Whether or not to keep the fitted values and model (default to TRUE, set to FALSE to reduce memory footprint).
`predict`	**[logical]** Whether or not to generate the smoothed predictions (necessary to fit the final SPM model, default to TRUE).

Value

The updated object.

`method_func` *Accessing OR replacing* `discretization_method` *model elements*

Description

All methods described here allow to access the elements of contained in objects of class discretization_method.

Usage

```
method_func(sspm_object)

## S4 method for signature 'discretization_method'
method_func(sspm_object)

method_func(object) <- value

## S4 replacement method for signature 'discretization_method'
method_func(object) <- value

## S4 method for signature 'discretization_method'
spm_name(sspm_object)

## S4 replacement method for signature 'discretization_method'
spm_name(object) <- value
```

Arguments

`sspm_object`	**[discretization_method]** An object of class discretization_method.
`object`	object from which to extract element(s) or in which to replace element(s).
`value`	typically an array-like R object of a similar class as `x`.

Value

The object in the required slot.

Examples

```
## Not run:
method_func(sspm_object)
...

## End(Not run)
```

`plot` *Plot* `sspm` *objects*

Description

Plot methods for a range of sspm objects.

Usage

```
## S4 method for signature 'sspm_boundary,missing'
plot(x, y, ...)

## S4 method for signature 'sspm_dataset,missing'
plot(
  x,
  y,
  ...,
  var = NULL,
  use_sf = FALSE,
  page = "first",
  nrow = 2,
  ncol = 2,
  log = FALSE,
  scales = "fixed"
)

## S4 method for signature 'sspm_fit,missing'
plot(
  x,
  y,
  ...,
  train_test = FALSE,
  biomass = NULL,
  next_ts = FALSE,
  aggregate = FALSE,
  interval = FALSE,
  biomass_origin = NULL,
  use_sf = FALSE,
  page = "first",
  nrow = 2,
  ncol = 2,
  log = FALSE,
  scales = "fixed"
)
```

Arguments

`x`	**[sspm_...]** An object from this package.
`y`	NOT USED (from generic).
`...`	NOT USED (from generic).
`var`	**[character]** (For sspm_dataset) Variable to plot.
`use_sf`	**[logical]** Whether to produce a spatial plot.
`page`	The page to draw
`nrow`	Number of rows per page

`ncol`	Number of columns per page
`log`	**[logical]** For productivity, whether to plot log productivity, (default to FALSE) for others, whether to plot on a log scale (default to TRUE).
`scales`	Are scales shared across all facets (the default, `"fixed"`), or do they vary across rows (`"free_x"`), columns (`"free_y"`), or both rows and columns (`"free"`)?
`train_test`	**[logical]** (For sspm_fit) Whether to plot a train/test pair plot.
`biomass`	**[character]** (For sspm_fit) The biomass variable for predictions.
`next_ts`	**[logical]** (For sspm_fit) Whether to plot a predictions for next timestep.
`aggregate`	**[logical]** (For sspm_fit) For biomass predictions only, whether to aggregate the data to the boundary level. Default to FALSE.
`interval`	**[logical]** (For sspm_fit) Whether to plot CI and Pi intervals.
`biomass_origin`	**[character]** Biomass variable to plot (from original dataset, optionnal).

Value

A ggplot2 plot object.

Examples

```
## Not run:
# To plot a boundary object and visualize patches/points
plot(sspm_boundary)
# To plot a dataset variable
plot(biomass_smooth, var = "weight_per_km2", log = FALSE)
plot(biomass_smooth, var = "weight_per_km2", use_sf = TRUE)
# To plot a fitted model
# Test-train plot
plot(sspm_model_fit, train_test = TRUE, scales = "free")
# Timeseries plot
plot(sspm_model_fit, log = T, scales = 'free')
plot(sspm_model_fit, log = T, use_sf = TRUE)
plot(sspm_model_fit, biomass = "weight_per_km2_borealis",  scales = "free")
plot(sspm_model_fit, biomass = "weight_per_km2_borealis", use_sf = TRUE)
plot(sspm_model_fit, biomass = "weight_per_km2_borealis",
     next_ts = TRUE, aggregate = TRUE, scales = "free", interval = T)

## End(Not run)
```

`plot_trajectory` *Plot trajectory of exploitation rate with biomass*

Description

Provides a trajectory plot for exploitation rate (ER), defined as the ratio between catch of a given year and fishable biomass from the previous year, and that fishable biomass.

Usage

```
plot_trajectory(sspm_object, dataset, biomass, catch, ...)

## S4 method for signature 'sspm_fit,sspm_dataset,character,character'
plot_trajectory(sspm_object, dataset, biomass, catch, dataset_biomass = NULL)
```

Arguments

`sspm_object`	**[sspm_dataset or sspm_fit]** The dataset object.
`dataset`	**[sspm_dataset]** Corresponding biomass dataset.
`biomass`	**[character]** Biomass variable for plotting.
`catch`	**[character]** Catch variable for plotting
`...`	More arguments passed onto `fun`
`dataset_biomass`	**[character]** Biomass variable for plotting in the biomass dataset if the variable name is different there.

Value

A ggplot2 plot object.

Examples

```
## Not run:
plot_trajectory(sspm_fit, borealis_dataset_fitted,
                "weight_per_km2", "catch")

## End(Not run)
```

`predator_simulated` *Simulated predator data*

Description

Simulated predator data for test and practice.

Usage

```
predator_simulated
```

Format

A data frame:

year_f Year as a factor

sfa SFA ID number

weight_per_km2 Simualated biomass in kg per km2

lon_dec Longitude

lat_dec Latitude

row Row ID

uniqueID Unique ID for simulated observation

`predict` *Predict with a SPM model*

Description

Predict using a fitted SPM model on the whole data or on new data

Usage

```
## S4 method for signature 'sspm_fit'
predict(
  object,
  new_data = NULL,
  biomass = NULL,
  aggregate = FALSE,
  interval = FALSE,
  next_ts = FALSE,
  type = "response"
)

## S4 method for signature 'sspm_dataset'
predict(object, new_data = NULL, discrete = TRUE, type = "response")
```

Arguments

`object`	**[sspm_fit]** Fit object to predict from.
`new_data`	**[data.frame]** New data to predict with.
`biomass`	**[character]** Biomass variable.
`aggregate`	**[logical]** For biomass predictions only, whether to aggregate the data to the boundary level. Default to FALSE.
`interval`	**[logical]** Whether or not to calculate confidence, and when possible, prediction intervals.
`next_ts`	**[logical]** For biomass, predict next timestep.

`type`	When this has the value `"link"` (default) the linear predictor (possibly with associated standard errors) is returned. When `type="terms"` each component of the linear predictor is returned seperately (possibly with standard errors): this includes parametric model components, followed by each smooth component, but excludes any offset and any intercept. `type="iterms"` is the same, except that any standard errors returned for smooth components will include the uncertainty about the intercept/overall mean. When `type="response"` predictions on the scale of the response are returned (possibly with approximate standard errors). When `type="lpmatrix"` then a matrix is returned which yields the values of the linear predictor (minus any offset) when postmultiplied by the parameter vector (in this case `se.fit` is ignored). The latter option is most useful for getting variance estimates for quantities derived from the model: for example integrated quantities, or derivatives of smooths. A linear predictor matrix can also be used to implement approximate prediction outside `R` (see example code, below).
`discrete`	**[logical]** If `new_data` is NULL, whether to predict based on a discrete prediction matrix (default to TRUE).

Value

A `dataframe` of predictions.

Examples

```
## Not run:
# Predictions for a model fit (usually, productivity)
predict(sspm_model_fit)
# To get biomass predictions, provide the variable name
predict(sspm_model_fit, biomass = "weight_per_km2_borealis")
# To get the next timestep predictions
predict(sspm_model_fit, biomass = "weight_per_km2_borealis", next_ts = TRUE)

## End(Not run)
```

`predict_intervals` *GAM confidence and prediction intervals*

Description

Computes CI from posterior, and PI for Tweedie and scat gams.

Usage

```
predict_intervals(object_fit, new_data, n = 1000, CI = TRUE, PI = TRUE, ...)
```

Arguments

`object_fit`	**[gam OR bam]** The fit to use for predictions.
`new_data`	**[data.frame]** The data to predict onto.
`n`	**[numeric]** The number of simulations to run for parameters.
`CI`	**[logical]** Whether to compute the CI.
`PI`	**[logical]** Whether to compute the PI.
`...`	further arguments passed to the quantile function.

Value

A `data.frame` with intervals.

Examples

```
gam1 <- gam(cyl ~ mpg, data=mtcars, family = tw)
predict_intervals(gam1)
```

`raw_formula` *Accessing OR replacing* `sspm_formula` *model elements*

Description

All methods described here allow to access the elements of contained in objects of class sspm_formula.

Usage

```
raw_formula(sspm_object)

## S4 method for signature 'sspm_formula'
raw_formula(sspm_object)

raw_formula(object) <- value

## S4 replacement method for signature 'sspm_formula'
raw_formula(object) <- value

translated_formula(sspm_object)

## S4 method for signature 'sspm_formula'
translated_formula(sspm_object)

translated_formula(object) <- value

## S4 replacement method for signature 'sspm_formula'
```

```
translated_formula(object) <- value

dataset(sspm_object)

## S4 method for signature 'sspm_formula'
dataset(sspm_object)

dataset(object) <- value

## S4 replacement method for signature 'sspm_formula'
dataset(object) <- value

formula_vars(sspm_object)

## S4 method for signature 'sspm_formula'
formula_vars(sspm_object)

formula_vars(object) <- value

## S4 replacement method for signature 'sspm_formula'
formula_vars(object) <- value

formula_type(sspm_object)

## S4 method for signature 'sspm_formula'
formula_type(sspm_object)

formula_type(object) <- value

## S4 replacement method for signature 'sspm_formula'
formula_type(object) <- value

is_fitted(sspm_object)

## S4 method for signature 'sspm_formula'
is_fitted(sspm_object)

is_fitted(object) <- value

## S4 replacement method for signature 'sspm_formula'
is_fitted(object) <- value

spm_response(sspm_object)

## S4 method for signature 'sspm_formula'
spm_response(sspm_object)

spm_response(object) <- value
```

```
## S4 replacement method for signature 'sspm_formula'
spm_response(object) <- value

spm_lagged_vars(sspm_object)

## S4 method for signature 'sspm_formula'
spm_lagged_vars(sspm_object)

spm_lagged_vars(object) <- value

## S4 replacement method for signature 'sspm_formula'
spm_lagged_vars(object) <- value
```

Arguments

`sspm_object`	**[sspm_formula]** An object of class sspm_formula.
`object`	object from which to extract element(s) or in which to replace element(s).
`value`	typically an array-like R object of a similar class as `x`.

Value

The object in the required slot.

Examples

```
## Not run:
translated_formula(sspm_object)
...

## End(Not run)
```

`sfa_boundaries` *SFA boundaries data*

Description

SFA boundaries.

Usage

```
sfa_boundaries
```

Format

A data frame and sf object:

sfa	SFA ID number
geometry	sf geometry

Source

https://www.dfo-mpo.gc.ca/fisheries-peches/ifmp-gmp/shrimp-crevette/shrimp-crevette-2018-002-eng.html

---

`smooth_time` *sspm Smoothing functions*

Description

A full sspm formula contains calls to the smoothing terms `smooth_time()`, `smooth_space()`, `smooth_space_time()`.

Usage

```
smooth_time(
  data_frame,
  boundaries,
  time,
  type = "ICAR",
  k = NULL,
  bs = "re",
  xt = NA,
  is_spm = FALSE,
  ...
)

smooth_space(
  data_frame,
  boundaries,
  time,
  type = "ICAR",
  k = NULL,
  bs = "mrf",
  xt = NULL,
  is_spm = FALSE,
  ...
)

smooth_space_time(
  data_frame,
  boundaries,
  time,
  type = "ICAR",
  k = c(NA, 30),
  bs = c("re", "mrf"),
  xt = list(NA, NULL),
```

```
  is_spm = FALSE,
  ...
)

smooth_lag(
  var,
  data_frame,
  boundaries,
  time,
  type = "LINPRED",
  k = 5,
  m = 1,
  ...
)

## S4 method for signature 'sf,sspm_discrete_boundary'
smooth_time(
  data_frame,
  boundaries,
  time,
  type = "ICAR",
  k = NULL,
  bs = "re",
  xt = NA,
  is_spm = FALSE,
  ...
)

## S4 method for signature 'sf,sspm_discrete_boundary'
smooth_space(
  data_frame,
  boundaries,
  time,
  type = "ICAR",
  k = NULL,
  bs = "mrf",
  xt = NULL,
  is_spm = FALSE,
  ...
)

## S4 method for signature 'sf,sspm_discrete_boundary'
smooth_space_time(
  data_frame,
  boundaries,
  time,
  type = "ICAR",
  k = c(NA, 30),
```

```
  bs = c("re", "mrf"),
  xt = list(NA, NULL),
  is_spm = FALSE,
  ...
)

## S4 method for signature 'ANY,sf,sspm_discrete_boundary'
smooth_lag(
  var,
  data_frame,
  boundaries,
  time,
  type = "LINPRED",
  k = 5,
  m = 1,
  ...
)
```

Arguments

`data_frame`	**[sf data.frame]** The data.
`boundaries`	**[sspm_boundary]** An object of class sspm_discrete_boundary.
`time`	**[character]** The time column.
`type`	**[character]** Type of smooth, currently only "ICAR" is supported.
`k`	**[numeric]** Size of the smooths and/or size of the lag.
`bs`	a two letter character string indicating the (penalized) smoothing basis to use. (eg `"tp"` for thin plate regression spline, `"cr"` for cubic regression spline). see `smooth.terms` for an over view of what is available.
`xt`	Any extra information required to set up a particular basis. Used e.g. to set large data set handling behaviour for `"tp"` basis. If `xt$sumConv` exists and is `FALSE` then the summation convention for matrix arguments is turned off.
`is_spm`	Whether or not an SPM is being fitted (used internally)
`...`	a list of variables that are the covariates that this smooth is a function of. Transformations whose form depends on the values of the data are best avoided here: e.g. `s(log(x))` is fine, but `s(I(x/sd(x)))` is not (see `predict.gam`).
`var`	**[symbol]** Variable (only for smooth_lag).
`m`	The order of the penalty for this term (e.g. 2 for normal cubic spline penalty with 2nd derivatives when using default t.p.r.s basis). `NA` signals autoinitialization. Only some smooth classes use this. The `"ps"` class can use a 2 item array giving the basis and penalty order separately.

Value

A list of 2 lists:

- `args`, contains the arguments to be passed on to the mgcv smooths
- `vars`, contains variables relevant to the evaluation of the smooth.

Examples

```
## Not run:
# Not meant to be used directly
smooth_time(borealis_data, bounds_voronoi, time = "year")

## End(Not run)
```

`spm` *Fit an SPM model*

Description

Fit an spm model to a sspm object

Usage

```
spm(sspm_object, formula, ...)

## S4 method for signature 'sspm,missing'
spm(sspm_object, formula, ...)

## S4 method for signature 'sspm,formula'
spm(sspm_object, formula, ...)
```

Arguments

`sspm_object`	**[sspm_dataset]** An object of class sspm_dataset.
`formula`	**[formula]** A formula definition of the form response ~ smoothing_terms + ...
`...`	Arguments passed on to `mgcv::bam` `family` This is a family object specifying the distribution and link to use in fitting etc. See `glm` and `family` for more details. The extended families listed in `family.mgcv` can also be used. `data` A data frame or list containing the model response variable and covariates required by the formula. By default the variables are taken from `environment(formula)`: typically the environment from which `gam` is called. `weights` prior weights on the contribution of the data to the log likelihood. Note that a weight of 2, for example, is equivalent to having made exactly the same observation twice. If you want to reweight the contributions of each datum without changing the overall magnitude of the log likelihood, then you should normalize the weights (e.g. `weights <- weights/mean(weights)`). `subset` an optional vector specifying a subset of observations to be used in the fitting process. `na.action` a function which indicates what should happen when the data contain ‘NA’s. The default is set by the ‘na.action’ setting of ‘options’, and is ‘na.fail’ if that is unset. The “factory-fresh” default is ‘na.omit’.

`offset` Can be used to supply a model offset for use in fitting. Note that this offset will always be completely ignored when predicting, unlike an offset included in `formula` (this used to conform to the behaviour of `lm` and `glm`).

`method` The smoothing parameter estimation method. `"GCV.Cp"` to use GCV for unknown scale parameter and Mallows' Cp/UBRE/AIC for known scale. `"GACV.Cp"` is equivalent, but using GACV in place of GCV. `"REML"` for REML estimation, including of unknown scale, `"P-REML"` for REML estimation, but using a Pearson estimate of the scale. `"ML"` and `"P-ML"` are similar, but using maximum likelihood in place of REML. Default `"fREML"` uses fast REML computation.

`control` A list of fit control parameters to replace defaults returned by `gam.control`. Any control parameters not supplied stay at their default values.

`select` Should selection penalties be added to the smooth effects, so that they can in principle be penalized out of the model? See `gamma` to increase penalization. Has the side effect that smooths no longer have a fixed effect component (improper prior from a Bayesian perspective) allowing REML comparison of models with the same fixed effect structure.

`scale` If this is positive then it is taken as the known scale parameter. Negative signals that the scale paraemter is unknown. 0 signals that the scale parameter is 1 for Poisson and binomial and unknown otherwise. Note that (RE)ML methods can only work with scale parameter 1 for the Poisson and binomial cases.

`gamma` Increase above 1 to force smoother fits. `gamma` is used to multiply the effective degrees of freedom in the GCV/UBRE/AIC score (so `log(n)/2` is BIC like). `n/gamma` can be viewed as an effective sample size, which allows it to play a similar role for RE/ML smoothing parameter estimation.

`knots` this is an optional list containing user specified knot values to be used for basis construction. For most bases the user simply supplies the knots to be used, which must match up with the `k` value supplied (note that the number of knots is not always just `k`). See `tprs` for what happens in the `"tp"`/`"ts"` case. Different terms can use different numbers of knots, unless they share a covariate.

`sp` A vector of smoothing parameters can be provided here. Smoothing parameters must be supplied in the order that the smooth terms appear in the model formula. Negative elements indicate that the parameter should be estimated, and hence a mixture of fixed and estimated parameters is possible. If smooths share smoothing parameters then `length(sp)` must correspond to the number of underlying smoothing parameters.

`min.sp` Lower bounds can be supplied for the smoothing parameters. Note that if this option is used then the smoothing parameters `full.sp`, in the returned object, will need to be added to what is supplied here to get the smoothing parameters actually multiplying the penalties. `length(min.sp)` should always be the same as the total number of penalties (so it may be longer than `sp`, if smooths share smoothing parameters).

`paraPen` optional list specifying any penalties to be applied to parametric model terms. `gam.models` explains more.

`chunk.size` The model matrix is created in chunks of this size, rather than ever being formed whole. Reset to `4*p` if `chunk.size` < `4*p` where `p` is the number of coefficients.

`rho` An AR1 error model can be used for the residuals (based on dataframe order), of Gaussian-identity link models. This is the AR1 correlation parameter. Standardized residuals (approximately uncorrelated under correct model) returned in `std.rsd` if non zero. Also usable with other models when `discrete=TRUE`, in which case the AR model is applied to the working residuals and corresponds to a GEE approximation.

`AR.start` logical variable of same length as data, `TRUE` at first observation of an independent section of AR1 correlation. Very first observation in data frame does not need this. If `NULL` then there are no breaks in AR1 correlaion.

`discrete` with `method="fREML"` it is possible to discretize covariates for storage and efficiency reasons. If `discrete` is `TRUE`, a number or a vector of numbers for each smoother term, then discretization happens. If numbers are supplied they give the number of discretization bins.

`cluster` `bam` can compute the computationally dominant QR decomposition in parallel using parLapply from the `parallel` package, if it is supplied with a cluster on which to do this (a cluster here can be some cores of a single machine). See details and example code.

`nthreads` Number of threads to use for non-cluster computation (e.g. combining results from cluster nodes). If `NA` set to `max(1,length(cluster))`. See details.

`gc.level` to keep the memory footprint down, it can help to call the garbage collector often, but this takes a substatial amount of time. Setting this to zero means that garbage collection only happens when R decides it should. Setting to 2 gives frequent garbage collection. 1 is in between. Not as much of a problem as it used to be.

`use.chol` By default `bam` uses a very stable QR update approach to obtaining the QR decomposition of the model matrix. For well conditioned models an alternative accumulates the crossproduct of the model matrix and then finds its Choleski decomposition, at the end. This is somewhat more efficient, computationally.

`samfrac` For very large sample size Generalized additive models the number of iterations needed for the model fit can be reduced by first fitting a model to a random sample of the data, and using the results to supply starting values. This initial fit is run with sloppy convergence tolerances, so is typically very low cost. `samfrac` is the sampling fraction to use. 0.1 is often reasonable.

`coef` initial values for model coefficients

`drop.unused.levels` by default unused levels are dropped from factors before fitting. For some smooths involving factor variables you might want to turn this off. Only do so if you know what you are doing.

`G` if not `NULL` then this should be the object returned by a previous call to `bam` with `fit=FALSE`. Causes all other arguments to be ignored except `sp`, `chunk.size`, `gamma`,`nthreads`, `cluster`, `rho`, `gc.level`, `samfrac`, `use.chol`, `method` and `scale` (if >0).

`fit` if `FALSE` then the model is set up for fitting but not estimated, and an object is returned, suitable for passing as the `G` argument to `bam`.

`drop.intercept` Set to `TRUE` to force the model to really not have the a constant in the parametric model part, even with factor variables present.

Value

An object of type `sspm_fit`.

Examples

```
## Not run:
sspm_model_fit <- sspm_model %>%
    spm(log_productivity ~ sfa +
    weight_per_km2_all_predators_lag_1 +
    smooth_space(by = weight_per_km2_borealis_with_catch) +
    smooth_space(),
    family = mgcv::scat)

## End(Not run)
```

`spm_aggregate` *Aggregate a dataset or fit data variable based on a boundary*

Description

Aggregate the data contained in a dataset or fit based on the discretized boundaries, using a function and a filling value.

Usage

```
spm_aggregate(
  sspm_object,
  boundaries,
  level = "patch",
  type = "data",
  variable,
  fun,
  group_by = "spacetime",
  fill = FALSE,
  apply_to_df = FALSE,
  ...
)

## S4 method for signature 'sspm_dataset,missing'
spm_aggregate(
  sspm_object,
  boundaries,
  level = "patch",
```

```
  type = "data",
  variable,
  fun,
  group_by = "spacetime",
  fill = FALSE,
  apply_to_df = FALSE,
  ...
)

## S4 method for signature 'sspm_dataset,sspm_discrete_boundary'
spm_aggregate(
  sspm_object,
  boundaries,
  level = "patch",
  type = "data",
  variable,
  fun,
  group_by = "spacetime",
  fill = FALSE,
  apply_to_df = FALSE,
  ...
)

## S4 method for signature 'sspm_fit,ANY'
spm_aggregate(
  sspm_object,
  boundaries,
  level = "patch",
  type = "data",
  variable,
  fun,
  group_by = "spacetime",
  fill = FALSE,
  apply_to_df = FALSE,
  ...
)
```

Arguments

`sspm_object`	**[sspm_dataset or sspm_fit]** The dataset object.
`boundaries`	**[sspm_discrete_boundary]** The boundaries object (optionnal).
`level`	**[character]** The aggregation level, "patch" or "boundary".
`type`	**[character]** The targeted type of aggregation, one of "data" for base data or "smoothed" for smoothed data.
`variable`	**[character]** Variable to aggregate (ignored in case `apply_to_df` is `TRUE`).
`fun`	**[function]** Function to use to aggregate data.
`group_by`	**[character]** One of `time`, `space` and `spacetime`.

`fill`	**[logical OR numeric OR function]** Whether to complete the incomplete cases, default to `FALSE` for no completion.
`apply_to_df`	**[logical]** Wether `fun` applied to the data frame group or to `variable`, default to `FALSE`.
`...`	More arguments passed onto `fun`

Value

Updated `sspm_dataset` or `sspm_fit`.

Examples

```
## Not run:
spm_aggregate(sspm_object = catch,
              boundaries = spm_boundaries(biomass),
              variable = catch_variable,
              fun = fun, group_by = group_by,
              fill = fill, apply_to_df = apply_to_df,
              na.rm = TRUE, ...)

## End(Not run)
```

`spm_aggregate_catch` *Update biomass value from catch adta*

Description

Aggregate the catch data contained in a catch dataset and update the biomass dataset with the subtracted catch.

Usage

```
spm_aggregate_catch(
  biomass,
  catch,
  biomass_variable,
  catch_variable,
  corrections = NULL,
  fun = sum,
  group_by = "spacetime",
  fill,
  apply_to_df = FALSE,
  ...
)

## S4 method for signature 'sspm_dataset,sspm_dataset,character,character'
spm_aggregate_catch(
```

```
    biomass,
    catch,
    biomass_variable,
    catch_variable,
    corrections = NULL,
    fun = sum,
    group_by = "spacetime",
    fill,
    apply_to_df = FALSE,
    ...
  )
```

Arguments

`biomass`	**[sspm_dataset (smoothed)]** The dataset containing the biomass variable.
`catch`	**[sspm_dataset]** The dataset containing the catch variable.
`biomass_variable`	**[character]** The biomass variab of `biomass`.
`catch_variable`	**[character]** The catch column of `catch`.
`corrections`	**[data.frame]** Optional landings corrections.
`fun`	**[function]** Function to use to aggregate data.
`group_by`	**[character]** One of `time`, `space` and `spacetime`.
`fill`	**[logical OR numeric OR function]** Whether to complete the incomplete cases, default to `FALSE` for no completion.
`apply_to_df`	**[logical]** Wether `fun` applied to the data frame group or to `variable`, default to `FALSE`.
`...`	More arguments passed onto `fun`

Value

Updated `sspm_dataset`.

Examples

```
## Not run:
spm_aggregate_catch(biomass = biomass_smooth, catch = catch_dataset,
                    biomass_variable = "weight_per_km2",
                    catch_variable = "catch",
                    fill = mean)

## End(Not run)
```

`spm_as_boundary` *Create a sspm_boundary object*

Description

Create a sspm_boundary object.

Usage

```
spm_as_boundary(
  boundaries,
  boundary,
  patches = NULL,
  points = NULL,
  boundary_area = NULL,
  patch_area = NULL
)

## S4 method for signature 'missing,ANY,ANY,ANY'
spm_as_boundary(
  boundaries,
  boundary,
  patches = NULL,
  points = NULL,
  boundary_area = NULL,
  patch_area = NULL
)

## S4 method for signature 'ANY,missing,ANY,ANY'
spm_as_boundary(
  boundaries,
  boundary,
  patches = NULL,
  points = NULL,
  boundary_area = NULL,
  patch_area = NULL
)

## S4 method for signature 'sf,character,missing,missing'
spm_as_boundary(
  boundaries,
  boundary,
  patches = NULL,
  points = NULL,
  boundary_area = NULL,
  patch_area = NULL
)
```

```
## S4 method for signature 'sf,character,ANY,ANY'
spm_as_boundary(
  boundaries,
  boundary,
  patches = NULL,
  points = NULL,
  boundary_area = NULL,
  patch_area = NULL
)
```

Arguments

`boundaries`	**[sf]** The sf object to cast.
`boundary`	**[character]** The column that contains the possible subdivisions of the boundaries.
`patches`	**[sf]** Patches resulting from discretization.
`points`	**[sf]** Sample points used for discretization.
`boundary_area`	**[character]** The column that contains the area of the subdivisions (optional).
`patch_area`	**[character]** The column that contains the area of the patches (optional).

Value

An object of class sspm_boundary or sspm_discrete_boundary.

Examples

```
sfa_boundaries
bounds <- spm_as_boundary(boundaries = sfa_boundaries,
                          boundary = "sfa")
plot(bounds)
```

`spm_as_dataset` *Create a* `sspm_dataset` *dataset structure*

Description

This casts a `data.frame` or `sf` object into an object of class `sspm_dataset`.

Usage

```
spm_as_dataset(data, name, time, uniqueID, coords = NULL, ...)

## S4 method for signature 'data.frame,ANY,ANY,ANY,missingOrNULL'
spm_as_dataset(
```

```
  data,
  name,
  time,
  uniqueID,
  coords,
  crs = NULL,
  boundaries = NULL,
  biomass = NULL,
  density = NULL,
  biomass_units = NULL,
  density_units = NULL
)

## S4 method for signature 'data.frame,ANY,ANY,ANY,list'
spm_as_dataset(
  data,
  name,
  time,
  uniqueID,
  coords,
  crs = NULL,
  boundaries = NULL,
  biomass = NULL,
  density = NULL,
  biomass_units = "kg",
  density_units = "kg/km^2"
)

## S4 method for signature 'data.frame,ANY,ANY,ANY,character'
spm_as_dataset(
  data,
  name,
  time,
  uniqueID,
  coords,
  crs = NULL,
  boundaries = NULL,
  biomass = NULL,
  density = NULL,
  biomass_units = "kg",
  density_units = "kg/km^2"
)

## S4 method for signature 'sf,ANY,ANY,ANY,ANY'
spm_as_dataset(
  data,
  name,
  time,
```

```
  uniqueID,
  coords,
  crs = NULL,
  boundaries = NULL,
  biomass = NULL,
  density = NULL,
  biomass_units = "kg",
  density_units = "kg/km^2"
)
```

Arguments

`data`	**[data.frame OR sf]** The dataset.
`name`	**[character]** The name of the dataset, default to "Biomass".
`time`	**[character]** The column of `data` for the temporal dimensions (i.e. year).
`uniqueID`	**[character]** The column of `data` that is unique for all rows of the data matrix.
`coords`	**[character]** The column of `data` for longitude and latitude of the observations.
`...`	Arguments passed onto methods.
`crs`	Coordinate reference system, passed onto st_as_sf.
`boundaries`	**[sspm_boundary]** An object of class sspm_discrete_boundary.
`biomass`	**[character]** Columns to be encoded as biomasses (required).
`density`	**[character]** Columns to be encoded as densities (optionnal).
`biomass_units`	**[character]** Units for biomass columns, default to "kg".
`density_units`	**[character]** Units for density columns, default to "kg/km^2".

Value

An object of class `sspm_dataset`.

Examples

```
data(borealis_simulated, package = "sspm")
biomass_dataset <- spm_as_dataset(borealis_simulated, name = "borealis",
                                  density = "weight_per_km2",
                                  time = "year_f",
                                  coords = c('lon_dec','lat_dec'),
                                  uniqueID = "uniqueID")
biomass_dataset
```

`spm_boundaries,sspm_boundary-method`

Accessing OR replacing `sspm_boundary` *model elements*

Description

All methods described here allow to access the elements of contained in objects of class sspm_boundary.

Usage

```
## S4 method for signature 'sspm_boundary'
spm_boundaries(sspm_object)

## S4 replacement method for signature 'sspm_boundary'
spm_boundaries(object) <- value

spm_discret_method(sspm_object)

## S4 method for signature 'sspm_discrete_boundary'
spm_discret_method(sspm_object)

spm_discret_method(object) <- value

## S4 replacement method for signature 'sspm_discrete_boundary'
spm_discret_method(object) <- value

spm_patches(sspm_object)

## S4 method for signature 'sspm_discrete_boundary'
spm_patches(sspm_object)

spm_patches(object) <- value

## S4 replacement method for signature 'sspm_discrete_boundary'
spm_patches(object) <- value

spm_points(sspm_object)

## S4 method for signature 'sspm_discrete_boundary'
spm_points(sspm_object)

spm_points(object) <- value

## S4 replacement method for signature 'sspm_discrete_boundary'
spm_points(object) <- value

spm_boundary(sspm_object)
```

```
## S4 method for signature 'sspm_boundary'
spm_boundary(sspm_object)

spm_boundary(object) <- value

## S4 replacement method for signature 'sspm_boundary'
spm_boundary(object) <- value

spm_boundary_area(sspm_object)

## S4 method for signature 'sspm_boundary'
spm_boundary_area(sspm_object)

spm_boundary_area(object) <- value

## S4 replacement method for signature 'sspm_boundary'
spm_boundary_area(object) <- value

spm_patches_area(sspm_object)

## S4 method for signature 'sspm_discrete_boundary'
spm_patches_area(sspm_object)

spm_patches_area(object) <- value

## S4 replacement method for signature 'sspm_discrete_boundary'
spm_patches_area(object) <- value
```

Arguments

`sspm_object`	**[sspm_boundary]** An object of class sspm_boundary.
`object`	object from which to extract element(s) or in which to replace element(s).
`value`	typically an array-like R object of a similar class as `x`.

Value

The object in the required slot.

Examples

```
## Not run:
spm_boundaries(sspm_object)
...

## End(Not run)
```

spm_data *Accessing OR replacing* `sspm_dataset` *model elements*

Description

All methods described here allow to access the elements of contained in objects of class sspm_dataset.

Usage

```
spm_data(sspm_object)

## S4 method for signature 'sspm_dataset'
spm_data(sspm_object)

spm_data(object) <- value

## S4 replacement method for signature 'sspm_dataset'
spm_data(object) <- value

## S4 method for signature 'sspm_dataset'
spm_name(sspm_object)

## S4 replacement method for signature 'sspm_dataset'
spm_name(object) <- value

## S4 method for signature 'sspm_dataset'
spm_unique_ID(sspm_object)

## S4 replacement method for signature 'sspm_dataset'
spm_unique_ID(object) <- value

spm_coords_col(sspm_object)

## S4 method for signature 'sspm_dataset'
spm_coords_col(sspm_object)

spm_coords_col(object) <- value

## S4 replacement method for signature 'sspm_dataset'
spm_coords_col(object) <- value

## S4 method for signature 'sspm_dataset'
spm_time(sspm_object)

## S4 replacement method for signature 'sspm_dataset'
spm_time(object) <- value
```

```
spm_biomass_vars(sspm_object)

## S4 method for signature 'sspm_dataset'
spm_biomass_vars(sspm_object)

spm_biomass_vars(object) <- value

## S4 replacement method for signature 'sspm_dataset'
spm_biomass_vars(object) <- value

spm_density_vars(sspm_object)

## S4 method for signature 'sspm_dataset'
spm_density_vars(sspm_object)

spm_density_vars(object) <- value

## S4 replacement method for signature 'sspm_dataset'
spm_density_vars(object) <- value

spm_formulas(sspm_object)

## S4 method for signature 'sspm_dataset'
spm_formulas(sspm_object)

spm_formulas(object) <- value

## S4 replacement method for signature 'sspm_dataset'
spm_formulas(object) <- value

## S4 method for signature 'sspm_dataset'
spm_smoothed_data(sspm_object)

## S4 replacement method for signature 'sspm_dataset'
spm_smoothed_data(object) <- value

spm_smoothed_fit(sspm_object)

## S4 method for signature 'sspm_dataset'
spm_smoothed_fit(sspm_object)

spm_smoothed_fit(object) <- value

## S4 replacement method for signature 'sspm_dataset'
spm_smoothed_fit(object) <- value

spm_smoothed_vars(sspm_object)
```

```
## S4 method for signature 'sspm_dataset'
spm_smoothed_vars(sspm_object)

spm_smoothed_vars(object) <- value

## S4 replacement method for signature 'sspm_dataset'
spm_smoothed_vars(object) <- value

is_mapped(sspm_object)

## S4 method for signature 'sspm_dataset'
is_mapped(sspm_object)

is_mapped(object) <- value

## S4 replacement method for signature 'sspm_dataset'
is_mapped(object) <- value

## S4 method for signature 'sspm_dataset'
spm_boundaries(sspm_object)

## S4 replacement method for signature 'sspm_dataset'
spm_boundaries(object) <- value
```

Arguments

`sspm_object`	**[sspm_dataset]** An object of class sspm_dataset.
`object`	object from which to extract element(s) or in which to replace element(s).
`value`	typically an array-like R object of a similar class as `x`.

Value

The object in the required slot.

Examples

```
## Not run:
spm_data(sspm_object)
...

## End(Not run)
```

`spm_discretize` *Discretize a* `sspm` *model object*

Description

Discretize a sspm model object with a function from a discretization_method object class.

Usage

```
spm_discretize(boundary_object, method = "tesselate_voronoi", with = NULL, ...)

## S4 method for signature 'sspm_boundary,missing,ANY'
spm_discretize(boundary_object, method = "tesselate_voronoi", with = NULL, ...)

## S4 method for signature 'sspm_boundary,ANY,missing'
spm_discretize(boundary_object, method = "tesselate_voronoi", with = NULL, ...)

## S4 method for signature 'sspm_boundary,character,ANY'
spm_discretize(boundary_object, method = "tesselate_voronoi", with = NULL, ...)

## S4 method for signature 'sspm_boundary,`function`,ANY'
spm_discretize(boundary_object, method = "tesselate_voronoi", with = NULL, ...)

## S4 method for signature 'sspm_boundary,discretization_method,ANY'
spm_discretize(boundary_object, method = "tesselate_voronoi", with = NULL, ...)

## S4 method for signature 'sspm_discrete_boundary,ANY,ANY'
spm_discretize(boundary_object, method = "tesselate_voronoi", with = NULL, ...)
```

Arguments

`boundary_object`	**[sspm]** An object of class sspm_boundary.
`method`	**[character OR method]** Either a `character` from the list of available methods (see spm_methods for the list) **OR** an object of class discretization_method.
`with`	**[sspm_dataset OR sf]** Either an object of class sspm_dataset or a set of custom points.
`...`	**[named list]** Further arguments to be passed onto the function used in `method`.

Details

Custom discretization functions can be written. The function must:

1. Accept at least 1 argument: **boundaries** (the `sf` boundary object), and optionnaly **with** (can be NULL) a separate object to be used for discretization and **boundary**, the boundary column of **boundaries** (these last 2 arguments are passed and connot be overwritten but could be ignored).
2. Returns a named list with 2 elements: `patches`. an `sf` object that stores the discretized polygons, and `points`, an `sf` object that stores the points that were used for discretization.

Value

An object of class sspm_discrete_boundary (the updated and discretized `sspm` object given as input).

Examples

```
## Not run:
# Voronoi tesselation
bounds_voronoi <- bounds %>%
  spm_discretize(method = "tesselate_voronoi",
                 with = biomass_dataset,
                 nb_samples = 10)

# Custom method
custom_func <- function(boundaries, ...){
  args <- list(...)
  # Can access passed arguments with args$arg_name
  # Do your custom discretization
  # Careful: must return sf objects!
  return(list(patches = c(),
              points = c())
         )
}

spm_discretize(boundary_object, method = custom_func)

## End(Not run)
```

`spm_lag` *Create lagged columns in a sspm smoothed data slot*

Description

This function is a wrapper around lag (note that not all arguments are supported). The default value for the lag is the mean of the series.

Usage

```
spm_lag(sspm_object, vars, n = 1, default = "mean", ...)

## S4 method for signature 'sspm'
spm_lag(sspm_object, vars, n = 1, default = "mean", ...)

## S4 method for signature 'sspm_fit'
spm_lag(sspm_object, vars, n = 1, default = "mean", ...)
```

Arguments

`sspm_object`	**[sspm_dataset]** An object of class sspm_dataset.
`vars`	**[character]** Names of the variables to lag.
`n`	Positive integer of length 1, giving the number of positions to lead or lag by
`default`	Value used for non-existent rows. Defaults to `NA`.

`...`	a list of variables that are the covariates that this smooth is a function of. Transformations whose form depends on the values of the data are best avoided here: e.g. `s(log(x))` is fine, but `s(I(x/sd(x)))` is not (see `predict.gam`).

Value

Updated `sspm_object`.

Examples

```
## Not run:
sspm_model <- sspm_model %>%
    spm_lag(vars = c("weight_per_km2_borealis_with_catch",
                     "weight_per_km2_all_predators"),
                     n = 1)

## End(Not run)
```

`spm_methods` *Get the list of available discretization methods*

Description

Currently, only one discretization method is supported: * `"tesselate_voronoi"` Voronoi tessellation using the function tesselate_voronoi.

Usage

```
spm_methods()
```

Details

You can create your own method (tutorial TBD).

Value

A `character vector` of all available discretization methods.

`spm_name` *Accessing OR replacing* `sspm` *model elements*

Description

All methods described here allow to access the elements of contained in objects of the different classes of the package.

Usage

```
spm_name(sspm_object)

spm_name(object) <- value

spm_datasets(sspm_object)

## S4 method for signature 'sspm'
spm_datasets(sspm_object)

spm_datasets(object) <- value

## S4 replacement method for signature 'sspm'
spm_datasets(object) <- value

spm_boundaries(sspm_object)

## S4 method for signature 'sspm'
spm_boundaries(sspm_object)

spm_boundaries(object) <- value

## S4 replacement method for signature 'sspm'
spm_boundaries(object) <- value

spm_smoothed_data(sspm_object)

## S4 method for signature 'sspm'
spm_smoothed_data(sspm_object)

spm_smoothed_data(object) <- value

## S4 replacement method for signature 'sspm'
spm_smoothed_data(object) <- value

spm_time(sspm_object)

## S4 method for signature 'sspm'
```

```
spm_time(sspm_object)

spm_time(object) <- value

## S4 replacement method for signature 'sspm'
spm_time(object) <- value

is_split(sspm_object)

## S4 method for signature 'sspm'
is_split(sspm_object)

is_split(object) <- value

## S4 replacement method for signature 'sspm'
is_split(object) <- value

spm_unique_ID(sspm_object)

## S4 method for signature 'sspm'
spm_unique_ID(sspm_object)

spm_unique_ID(object) <- value

## S4 replacement method for signature 'sspm'
spm_unique_ID(object) <- value
```

Arguments

`sspm_object`	**[sspm OR adjacent]** An object of class sspm or others derivative classes.
`object`	object from which to extract element(s) or in which to replace element(s).
`value`	typically an array-like R object of a similar class as `x`.

Value

The object in the required slot.

Examples

```
## Not run:
spm_name(sspm_object)
...

## End(Not run)
```

`spm_smooth` *Smooth a variable in a sspm dataset*

Description

With a formula, smooth a variable in a sspm dataset. See Details for more explanations.

Usage

```
spm_smooth(
  sspm_object,
  formula,
  boundaries,
  keep_fit = TRUE,
  predict = TRUE,
  ...
)

## S4 method for signature 'sspm_dataset,formula,sspm_discrete_boundary'
spm_smooth(
  sspm_object,
  formula,
  boundaries,
  keep_fit = TRUE,
  predict = TRUE,
  ...
)
```

Arguments

`sspm_object`	**[sspm_dataset]** An object of class sspm_dataset.
`formula`	**[formula]** A formula definition of the form response ~ smoothing_terms + ...
`boundaries`	**[sspm_boundary]** An object of class sspm_discrete_boundary.
`keep_fit`	**[logical]** Whether or not to keep the fitted values and model (default to TRUE, set to FALSE to reduce memory footprint).
`predict`	**[logical]** Whether or not to generate the smoothed predictions (necessary to fit the final SPM model, default to TRUE).
`...`	Arguments passed on to `mgcv::bam` `family` This is a family object specifying the distribution and link to use in fitting etc. See `glm` and `family` for more details. The extended families listed in `family.mgcv` can also be used. `data` A data frame or list containing the model response variable and covariates required by the formula. By default the variables are taken from `environment(formula)`: typically the environment from which `gam` is called.

`weights` prior weights on the contribution of the data to the log likelihood. Note that a weight of 2, for example, is equivalent to having made exactly the same observation twice. If you want to reweight the contributions of each datum without changing the overall magnitude of the log likelihood, then you should normalize the weights (e.g. `weights <- weights/mean(weights)`).

`subset` an optional vector specifying a subset of observations to be used in the fitting process.

`na.action` a function which indicates what should happen when the data contain 'NA's. The default is set by the 'na.action' setting of 'options', and is 'na.fail' if that is unset. The "factory-fresh" default is 'na.omit'.

`offset` Can be used to supply a model offset for use in fitting. Note that this offset will always be completely ignored when predicting, unlike an offset included in `formula` (this used to conform to the behaviour of `lm` and `glm`).

`method` The smoothing parameter estimation method. `"GCV.Cp"` to use GCV for unknown scale parameter and Mallows' Cp/UBRE/AIC for known scale. `"GACV.Cp"` is equivalent, but using GACV in place of GCV. `"REML"` for REML estimation, including of unknown scale, `"P-REML"` for REML estimation, but using a Pearson estimate of the scale. `"ML"` and `"P-ML"` are similar, but using maximum likelihood in place of REML. Default `"fREML"` uses fast REML computation.

`control` A list of fit control parameters to replace defaults returned by `gam.control`. Any control parameters not supplied stay at their default values.

`select` Should selection penalties be added to the smooth effects, so that they can in principle be penalized out of the model? See `gamma` to increase penalization. Has the side effect that smooths no longer have a fixed effect component (improper prior from a Bayesian perspective) allowing REML comparison of models with the same fixed effect structure.

`scale` If this is positive then it is taken as the known scale parameter. Negative signals that the scale paraemter is unknown. 0 signals that the scale parameter is 1 for Poisson and binomial and unknown otherwise. Note that (RE)ML methods can only work with scale parameter 1 for the Poisson and binomial cases.

`gamma` Increase above 1 to force smoother fits. `gamma` is used to multiply the effective degrees of freedom in the GCV/UBRE/AIC score (so `log(n)/2` is BIC like). `n/gamma` can be viewed as an effective sample size, which allows it to play a similar role for RE/ML smoothing parameter estimation.

`knots` this is an optional list containing user specified knot values to be used for basis construction. For most bases the user simply supplies the knots to be used, which must match up with the `k` value supplied (note that the number of knots is not always just `k`). See `tprs` for what happens in the `"tp"/"ts"` case. Different terms can use different numbers of knots, unless they share a covariate.

`sp` A vector of smoothing parameters can be provided here. Smoothing parameters must be supplied in the order that the smooth terms appear in the model formula. Negative elements indicate that the parameter should be estimated, and hence a mixture of fixed and estimated parameters is possible. If smooths share smoothing parameters then `length(sp)` must correspond to the number of underlying smoothing parameters.

`min.sp` Lower bounds can be supplied for the smoothing parameters. Note that if this option is used then the smoothing parameters `full.sp`, in the returned object, will need to be added to what is supplied here to get the smoothing parameters actually multiplying the penalties. `length(min.sp)` should always be the same as the total number of penalties (so it may be longer than `sp`, if smooths share smoothing parameters).

`paraPen` optional list specifying any penalties to be applied to parametric model terms. `gam.models` explains more.

`chunk.size` The model matrix is created in chunks of this size, rather than ever being formed whole. Reset to `4*p` if `chunk.size` < `4*p` where `p` is the number of coefficients.

`rho` An AR1 error model can be used for the residuals (based on dataframe order), of Gaussian-identity link models. This is the AR1 correlation parameter. Standardized residuals (approximately uncorrelated under correct model) returned in `std.rsd` if non zero. Also usable with other models when `discrete=TRUE`, in which case the AR model is applied to the working residuals and corresponds to a GEE approximation.

`AR.start` logical variable of same length as data, `TRUE` at first observation of an independent section of AR1 correlation. Very first observation in data frame does not need this. If `NULL` then there are no breaks in AR1 correlaion.

`discrete` with `method="fREML"` it is possible to discretize covariates for storage and efficiency reasons. If `discrete` is `TRUE`, a number or a vector of numbers for each smoother term, then discretization happens. If numbers are supplied they give the number of discretization bins.

`cluster` `bam` can compute the computationally dominant QR decomposition in parallel using parLapply from the `parallel` package, if it is supplied with a cluster on which to do this (a cluster here can be some cores of a single machine). See details and example code.

`nthreads` Number of threads to use for non-cluster computation (e.g. combining results from cluster nodes). If `NA` set to `max(1,length(cluster))`. See details.

`gc.level` to keep the memory footprint down, it can help to call the garbage collector often, but this takes a substatial amount of time. Setting this to zero means that garbage collection only happens when R decides it should. Setting to 2 gives frequent garbage collection. 1 is in between. Not as much of a problem as it used to be.

`use.chol` By default `bam` uses a very stable QR update approach to obtaining the QR decomposition of the model matrix. For well conditioned models an alternative accumulates the crossproduct of the model matrix and then finds its Choleski decomposition, at the end. This is somewhat more efficient, computationally.

`samfrac` For very large sample size Generalized additive models the number of iterations needed for the model fit can be reduced by first fitting a model to a random sample of the data, and using the results to supply starting values. This initial fit is run with sloppy convergence tolerances, so is typically very low cost. `samfrac` is the sampling fraction to use. 0.1 is often reasonable.

`coef` initial values for model coefficients `drop.unused.levels` by default unused levels are dropped from factors before fitting. For some smooths involving factor variables you might want to turn this off. Only do so if you know what you are doing.

`G` if not `NULL` then this should be the object returned by a previous call to `bam` with `fit=FALSE`. Causes all other arguments to be ignored except `sp`, `chunk.size`, `gamma`,`nthreads`, `cluster`, `rho`, `gc.level`, `samfrac`, `use.chol`, `method` and `scale` (if >0).

`fit` if `FALSE` then the model is set up for fitting but not estimated, and an object is returned, suitable for passing as the `G` argument to `bam`.

`drop.intercept` Set to `TRUE` to force the model to really not have the a constant in the parametric model part, even with factor variables present.

Details

This functions allows to specify a model formula for a given discrete sspm object. The formula makes use of specific smoothing terms `smooth_time()`, `smooth_space()`, `smooth_space_time()`. The formula can also contain fixed effects and custom smooths, and can make use of specific smoothing terms `smooth_time()`, `smooth_space()`, `smooth_space_time()`.

Value

An updated sspm_dataset.

Examples

```
## Not run:
biomass_smooth <- biomass_dataset %>%
    spm_smooth(weight_per_km2 ~ sfa + smooth_time(by = sfa) +
                 smooth_space() +
                 smooth_space_time(),
                 boundaries = bounds_voronoi,
                 family = tw)

## End(Not run)
```

`spm_smooth_methods` *Get the list of available smoothing methods*

Description

Currently, only one smoothing method is supported: * `"ICAR"`: Intrinsic Conditional Auto-Regressive models. * `"LINPRED"`: LINear PREDictors (lag smooths).

Usage

```
spm_smooth_methods()
```

Value

A `character vector` of all available smoothing methods.

`spm_split` *Split data in test and train sets*

Description

Split data before fitting spm (WIP).

Usage

```
spm_split(sspm_object, ...)

## S4 method for signature 'sspm'
spm_split(sspm_object, ...)
```

Arguments

`sspm_object`	**[sspm]** An object of class sspm.
`...`	**[expression]** Expression to evaluate to split data.

Value

The updated sspm object.

Examples

```
## Not run:
sspm_model <- sspm_model %>%
    spm_split(year_f %in% c(1990:2017))

## End(Not run)
```

`spm_unique_ID,sspm_fit-method` *Accessing OR replacing* `sspm_fit` *model elements*

Description

All methods described here allow to access the elements of contained in objects of class sspm_fit.

Usage

```
## S4 method for signature 'sspm_fit'
spm_unique_ID(sspm_object)

## S4 replacement method for signature 'sspm_fit'
spm_unique_ID(object) <- value

## S4 method for signature 'sspm_fit'
spm_time(sspm_object)

## S4 replacement method for signature 'sspm_fit'
spm_time(object) <- value

## S4 method for signature 'sspm_fit'
spm_formulas(sspm_object)

## S4 replacement method for signature 'sspm_fit'
spm_formulas(object) <- value

## S4 method for signature 'sspm_fit'
spm_smoothed_data(sspm_object)

## S4 replacement method for signature 'sspm_fit'
spm_smoothed_data(object) <- value

spm_get_fit(sspm_object)

## S4 method for signature 'sspm_fit'
spm_get_fit(sspm_object)

spm_get_fit(object) <- value

## S4 replacement method for signature 'sspm_fit'
spm_get_fit(object) <- value

## S4 method for signature 'sspm_fit'
spm_boundaries(sspm_object)

## S4 replacement method for signature 'sspm_fit'
spm_boundaries(object) <- value

## S4 method for signature 'sspm_fit'
spm_boundary(sspm_object)

## S4 replacement method for signature 'sspm_fit'
spm_boundary(object) <- value
```

Arguments

`sspm_object`	**[sspm_fit]** An object of class sspm_fit.
`object`	object from which to extract element(s) or in which to replace element(s).
`value`	typically an array-like R object of a similar class as `x`.

Value

The object in the required slot.

Examples

```
## Not run:
spm_formulas(sspm_object)
...

## End(Not run)
```

`sspm` *Create a* `sspm` *model object*

Description

Create a sspm_model object.

Usage

```
sspm(biomass, predictors)

## S4 method for signature 'sspm_dataset,missing'
sspm(biomass, predictors)

## S4 method for signature 'sspm_dataset,sspm_dataset'
sspm(biomass, predictors)

## S4 method for signature 'sspm_dataset,list'
sspm(biomass, predictors)
```

Arguments

`biomass`	**[sspm_dataset (smoothed)]** The dataset containing the biomass variable.
`predictors`	**[list OF sspm_dataset (smoothed)]** The list of predictor datasets.

Value

An object of class sspm.

Examples

```
## Not run:
sspm_model <- sspm(biomass = biomass_smooth_w_catch,
                   predictors = predator_smooth)

## End(Not run)
```

`sspm-class` *sspm model class*

Description

The `sspm` model object, made from biomass, predictor and catch data.

Slots

`datasets` **[list]** List of sspm_dataset that define variables in the SPM model.

`time` **[character]** The column of `data` that represents the temporal dimension of the dataset.

`uniqueID` **[character]** The column of `datasets` that is unique for all rows of the data matrix.

`boundaries` **[sf]** Spatial boundaries (polygons).

`smoothed_data` **[ANY (sf)]** The smoothed data.

`smoothed_vars` **[character]** A vector storing the smoothed vars.

`is_split` **[logical]** Whether this object has been split into train/test sets.

`sspm_boundary-class` *sspm boundary structure*

Description

One of the first step in the `sspm` workflow is to create one or more object(s) of class `sspm_boundary` from an `sf` object.

Functions

- `sspm_discrete_boundary-class`: sspm_discrete_boundary

Slots

`boundaries` **[sf]** Spatial boundaries (polygons).

`boundary` **[character]** The column of `data` that represents the spatial boundaries.

`boundary_area` **[character]** The column of `data` that represents the area of spatial boundaries.

`method` **[discretization_method]** *(if discrete)* discretization method used.

`patches` **[sf]** *(if discrete)* Patches resulting from discretization.

`points` **[sf or NULL]** *(if discrete)* Sample points used for discretization.

`patches_area` **[character]** The column of `data` that represents the area of patches.

`sspm_dataset-class` *sspm dataset structure*

Description

One of the first step in the `sspm` workflow is to create one or more object(s) of class `sspm_dataset` from a `data.frame`, `tibble` or `sf` object.

Slots

`name` **[character]** The name of the dataset, default to "Biomass".

`data` **[data.frame OR sf OR tibble]** The dataset.

`biomass` **[character]** The biomass columns of `data`.

`density` **[character]** The biomass density columns of `data`.

`time` **[character]** The column of `data` that represents the temporal dimension of the dataset.

`coords` **[character]** The columns of `data` that represent the spatial dimension of the dataset: the two columns for longitude and latitude of the observations.

`uniqueID` **[character]** The column of `data` that is unique for all rows of the data matrix.

`boundaries` **[sspm_discrete_boundary]** Spatial boundaries (polygons).

`formulas` **[list]** List of sspm_formula objects that specifies the smoothed variables.

`smoothed_data` **[ANY (sf)]** The smoothed data.

`smoothed_vars` **[character]** A vector storing the smoothed vars.

`smoothed_fit` **[list]** The fit from smoothing the data

`is_mapped` **[logical]** Whether the dataset has been mapped to boundaries (used internally).

`sspm_fit-class` *sspm fit*

Description

The fit object for a sspm model

Slots

`smoothed_data` **[ANY (sf)]** The smoothed data.

`time` **[character]** The column of `smoothed_data` that represents the temporal dimension of the dataset.

`uniqueID` **[character]** The column of `smoothed_data` that is unique for all rows of the data matrix.

`formula` **[list]** The sspm_formula object that specifies the spm model.

`boundaries` **[sf]** Spatial boundaries (polygons).

`fit` **[bam]** The fit of the spm model.

`sspm_formula-class` *sspm formula object*

Description

This class is a wrapper around the `formula` class. It is not intended for users to directly manipulate and create new objects.

Slots

`raw_formula` **[formula]** The raw formula call

`translated_formula` **[formula]** The translated formula call ready to be evaluated.

`vars` **[list]** List of relevant variables for the evaluation of the different smooths.

`lag_vars` Smooth lag variables used for predictions

`response` **[charatcer]** The response variable in the formula.

`is_fitted` **[logical]** Whether this formula has already been fitted.

See Also

See the `mgcv` function for defining smooths: s().

`tesselate_voronoi` *Perform voronoi tesselation*

Description

Generates voronoi polygons by first performing stratified sampling across boundary polygons, then by running the voronoisation with st_voronoi().

Usage

```
tesselate_voronoi(
  boundaries,
  with,
  boundary = "sfa",
  sample_surface = FALSE,
  sample_points = TRUE,
  nb_samples = NULL,
  min_size = 1500,
  stratify = TRUE,
  seed = 1
)
```

Arguments

`boundaries`	**[sf]** The boundaries to be used.
`with`	**[sf]** A set of data points to use for voronoisation.
`boundary`	**[character]** The column in `boundaries` that is to be used for the stratified sampling.
`sample_surface`	**[logical]** Whether to sample the surfaces in `boundaries`, Default to `FALSE`.
`sample_points`	**[logical]** Whether to sample points from `with` or to take all points in `with`. Default to `TRUE`.
`nb_samples`	**[named character vector]** The number of samples to draw by boundary polygons (must bear the levels of `boundary` as names or be a single value to be applied to each level).
`min_size`	**[numeric]** The minimum size for a polygon above which it will be merged (in km2).
`stratify`	**[logical]** Whether the discretization happens within the boundaries or whether the whole area is to be used (default to TRUE).
`seed`	**[numeric]** Passed onto `set.seed()`, important for reproducibility of sampling.

Value

A named list with three elements (each an `sf` object): * `patches`, the voronoi polygons generated * `points`, the points used for the tessellation.

Examples

```
data(borealis_simulated, package = "sspm")
data(sfa_boundaries, package = "sspm")
tesselate_voronoi(sfa_boundaries, with = borealis, sample_surface = TRUE,
                  boundary = "sfa", nb_samples = 10)
```

`triangulate_delaunay` *Perform delaunay triangulation*

Description

Generates delaunay triangles with ct_triangulate().

Usage

```
triangulate_delaunay(
  boundaries,
  with = NULL,
  boundary = "sfa",
  sample_surface = FALSE,
  sample_points = FALSE,
```

```
    nb_samples = NULL,
    min_size = 1000,
    seed = 1,
    ...
  )
```

Arguments

`boundaries`	**[sf]** The boundaries to be used.
`with`	**[sf]** A set of data points to use for voronoisation.
`boundary`	**[character]** The column in `boundaries` that is to be used for the stratified sampling.
`sample_surface`	**[logical]** Whether to sample the surfaces in `boundaries`, Default to `FALSE`.
`sample_points`	**[logical]** Whether to sample points from `with` or to take all points in `with`. Default to `TRUE`.
`nb_samples`	**[named character vector]** The number of samples to draw by boundary polygons (must bear the levels of `boundary` as names or be a single value to be applied to each level).
`min_size`	**[numeric]** The minimum size for a triangle above which it will be merged (in km2).
`seed`	**[numeric]** Passed onto `set.seed()`, important for reproducibility of sampling.
`...`	Arguments passed on to `RTriangle::triangulate` `p` Planar straight line graph object; see `pslg`. `a` Maximum triangle area. If specified, triangles cannot be larger than this area. `q` Minimum triangle angle in degrees. `Y` If `TRUE` prohibits the insertion of Steiner points on the mesh boundary. `j` If `TRUE` jettisons vertices that are not part of the final triangulation from the output. `D` If `TRUE` produce a conforming Delaunay triangulation. This ensures that all the triangles in the mesh are truly Delaunay, and not merely constrained Delaunay. This option invokes Ruppert's original algorithm, which splits every subsegment whose diametral circle is encroached. It usually increases the number of vertices and triangles. `S` Specifies the maximum number of added Steiner points. `V` Verbosity level. Specify higher values for more detailed information about what the Triangle library is doing. `Q` If `TRUE` suppresses all explanation of what the Triangle library is doing, unless an error occurs.

Value

A named list with three elements (each an `sf` object): * `patches`, the voronoi polygons generated * `points`, the points used for the tessellation.

Examples

```
data(borealis_simulated, package = "sspm")
data(sfa_boundaries, package = "sspm")
triangulate_delaunay(sfa_boundaries, with = borealis, sample_surface = TRUE,
                     boundary = "sfa", nb_samples = 10)
```

`$,sspm_boundary-method` *Extract methods*

Description

WIP extract variables from sspm objects

Usage

```
## S4 method for signature 'sspm_boundary'
x$name

## S4 method for signature 'sspm_discrete_boundary'
x$name

## S4 method for signature 'sspm_dataset'
x$name

## S4 method for signature 'sspm'
x$name

## S4 method for signature 'sspm_fit'
x$name
```

Arguments

`x`	**[sspm_...]** An object from this package.
`name`	**[character]** The name of the column

Value

The `data.frame` matching the request.

Examples

```
## Not run:
sspm_boundary$lat
sspm_dataset$biomass

## End(Not run)
```

Index