Package 'rts2'

July 8, 2025

Title Real-Time Disease Surveillance

Version 0.8.3

Date 2025-07-08

Description Supports modelling real-time case data to facilitate the real-time surveillance of infectious diseases and other point phenomena. The package provides automated computational grid generation over an area of interest with methods to map covariates between geographies, model fitting including spatially aggregated case counts, and predictions and visualisation. Both Bayesian and maximum likelihood methods are provided. Log-Gaussian Cox Processes are described by Diggle et al. (2013) <doi:10.1214/13-STS441> and we provide both the low-rank approximation for Gaussian processes described by Solin and Särkkä (2020) <doi:10.1007/s11222-019-09886-w> and Riutort-Mayol et al (2023) <doi:10.1007/s11222-022-10167-2> and the nearest neighbour Gaussian process described by Datta et al (2016) <doi:10.1080/01621459.2015.1044091>. 'cmdstanr' can be downloaded at <https://mc-stan.org/cmdstanr/>.

License CC BY-SA 4.0

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Biarch true

Depends R (>= 3.5.0), sf (>= 1.0-14)

Imports methods, R6, Rcpp (>= 0.12.0), RcppParallel (>= 5.0.1), rstan (>= 2.26.0), rstantools (>= 2.1.1), lubridate (>= 1.9.0), stars (>= 0.6-1), raster (>= 3.6-1)

Suggests cmdstanr (>= 0.4.0), testthat

LinkingTo BH (>= 1.66.0), Rcpp (>= 0.12.0), RcppEigen (>= 0.3.3.3.0), RcppParallel (>= 5.0.1), rstan (>= 2.26.0), StanHeaders (>= 2.32.0), glmmrBase (>= 0.7.1), SparseChol (>= 0.2.2)

SystemRequirements GNU make

NeedsCompilation yes

Author Sam Watson [aut, cre] (ORCID: <https://orcid.org/0000-0002-8972-769X>)

Maintainer Sam Watson <s.i.watson@bham.ac.uk>

Repository CRAN

Date/Publication 2025-07-08 21:50:07 UTC

Contents

`rts2-package` *Real-Time Disease Surveillance*

Description

Supports modelling real-time case data to facilitate the real-time surveillance of infectious diseases and other point phenomena. The package provides automated computational grid generation over an area of interest with methods to map covariates between geographies, model fitting including spatially aggregated case counts, and predictions and visualisation. Both Bayesian and maximum likelihood methods are provided. Log-Gaussian Cox Processes are described by Diggle et al. (2013) <doi:10.1214/13-STS441> and we provide both the low-rank approximation for Gaussian processes described by Solin and Särkkä (2020) <doi:10.1007/s11222-019-09886-w> and Riutort-Mayol et al (2023) <doi:10.1007/s11222-022-10167-2> and the nearest neighbour Gaussian process described by Datta et al (2016) <doi:10.1080/01621459.2015.1044091>. 'cmdstanr' can be downloaded at <https://mc-stan.org/cmdstanr/>. `rts2` provides several estimators for the Log Gaussian Cox Process Model (LGCP). The LGCP is a stochastic Poisson model used for modelling case counts of phenomena of interest, and is particularly useful for predicting risk across an area of interest, such as in disease surveillance applications.

Workflow

Most of the functionality of the **rts2** package is provided by the `grid` class. The computational strategy for the LGCP is to divide up the area of interest into a regular grid and aggregate case counts within cells. For models with count data aggregated to an irregular set of polygons, such as census tracts, the latent surface is also modelled as a regular grid. A typical workflow using this package would be:

1. Create a new `grid` object, e.g. `g1 <- grid$new(poly, cellsize = 0.1)`. The class is initialized with either a single polygon describing the area of interest or a collection of polygons if spatially aggregated data are used. The **sf** package is used for all spatial data.
2. If the location (and times) of cases are available (i.e. the data are not spatially aggregated), then we map the points to the computational grid. The function create_points can generate point data in the correct **sf** format. The member function `points_to_grid` will then map these data to the grid. Counts can also be manually added to grid data. For region data, since the counts are assumed to be already aggregated, these must be manually provided by the user. The case counts must appear in columns with specific names. If there is only a single time period then the counts must be in a column named `y`. If there are multiple time periods then the counts must be in columns names `t1`, `t2`, `t3`,... Associated columns labelled `date1`, `date2`, etc. will permit use of some functionality regarding specific time intervals.
3. If any covariates are to be used for the modelling, then these can be mapped to the compuational grid using the function `add_covariates()`. Other functions, `add_time_indicators()` and `get_dow()` will also generate relevant temporal indicators where required. At a minimum we would recommend including a measure of population density.
4. Fit a model. There are multiple methods for model fitting, which are available through the member functions `lgcp_ml()` and `lgcp_bayes()` for maximum likelihood and Bayesian approaches, respectively. The results are stored internally and optionally returned as a `rtsFit` object.
5. Summarise the output. The main functions for summarising the output are `extract_preds()`, which will generate predictions of relative risk, incidence rate ratios, and predicted incidence, and `hotspots()`, which will estimate probabilities that these statistics exceed given thresholds. For spatially-aggregated data models, the relative risk applies to the grid, whereas rate ratios and predicted incidence applies to the areas.

6. Predictions can be visualised or aggregated to relevant geographies with the `plot()` and `aggregate()` functions.

Estimation methods and model specification

The **rts2** package provide several methods for estimating the model and drawing samples from the latent surface.

- Maximum Likelihood. We include stochastic maximum likelihood estimation methods including both Markov Chain Monte Carlo (MCMC) Maximum Likelihood and Stochastic Approximation Expectation Maximisation (SAEM). MCMC-ML can use Newton-Raphson, quasi-Newton, or derivative free methods to estimate the model parameters. Both algorithms have three steps: 1. Sample the random effects using MCMC; 2. Estimate the fixed effect parameters conditional on the sampled random effects; 3. Estimate the covariance parameters. The process is iterated until convergence. Stochastic maximum likelihood estimators are provided by the function `grid$lgcp_ml()`.
- Bayesian. We also include Bayesian estimation of the model using Stan via either **rstan** or **cmdstanr**, and allow both MCMC and Variational Bayes methods.

The LGCP can be computationally complex and scales poorly with sample size (number of grid cells and time periods), due to the large covariance matrix that must be inverted to estimate the covariance parameters. We offer several strategies and approximations for efficient model fitting:

- Gaussian Process Approximations. The package includes both Hilbert Space Gaussian Process (see Solin and Särkkä (2020) <doi:10.1007/s11222-019-09886-w> and Riutort-Mayol et al (2020) <arXiv:2004.11408>) and the Nearest Neighbour Gaussian Process (Datta et al (2016) <doi:10.1080/01621459.2015.1044091>).
- For spatio-temporal models we use a "spatial innovation" formulation of the spatio-temporal Gaussian process, for which the computational complexity is linear in the number of time periods.

Package development

The package is still in development and there may still be bugs and errors. While we do not expect the general user interface to change there may be changes to the underlying library as well as new additions and functionality.

Author(s)

Sam Watson [aut, cre] (ORCID: <https://orcid.org/0000-0002-8972-769X>)

Maintainer: Sam Watson <s.i.watson@bham.ac.uk>

References

Stan Development Team (2020). RStan: the R interface to Stan. R package version 2.21.2. https://mc-stan.org

birmingham_crime *Birmingham crime data*

Description

Counts of burglaries for the months of 2022 for the city of Birmingham, UK at the Middle-Layer Super Output Area.

Usage

```
birmingham_crime
```

Format

An object of class `sf` (inherits from `data.frame`) with 132 rows and 21 columns.

boundary *Boundary polygon for Birmingham, UK*

Description

A Boundary polygon describing the border of the city of Birmingham, UK.

Usage

```
boundary
```

Format

An object of class `sf` (inherits from `data.frame`) with 1 rows and 2 columns.

coef.rtsFit *Extracts fixed effect coefficients from a rtsFit object*

Description

Extracts the fitted fixed effect coefficients from an `rtsFit` object returned from a call of `rtsFit` or `LA` in the Model class.

Usage

```
## S3 method for class 'rtsFit'
coef(object, ...)
```

Arguments

object	An rtsFit model fit.
...	Further arguments passed from other methods

Value

A named vector.

---

confint.rtsFit *Fixed effect confidence intervals for a* rtsFit *object*

Description

Returns the computed confidence intervals for a rtsFit object.

Usage

```
## S3 method for class 'rtsFit'
confint(object, ...)
```

Arguments

object	A rtsFit object.
...	Further arguments passed from other methods

Value

A matrix (or vector) with columns giving lower and upper confidence limits for each parameter.

---

covariance.parameters *Extracts the estimates of the covariance parameters*

Description

Extracts the estimates of the covariance parameters an rtsFit object returned from call of lgcp_ml() or lgcp_bayes() in the grid class.

Usage

```
covariance.parameters(object)
```

Arguments

object	An mcml model fit.

Value

A matrix of dimension (number of fixed effects) x (number of MCMC samples). For Laplace approximation, the number of "samples" equals one.

`create_points` *Create sf object from point location data*

Description

Produces an sf object with location and time of cases from a data frame

Usage

```
create_points(
  data,
  pos_vars = c("lat", "long"),
  t_var = NULL,
  format = "%Y-%m-%d",
  verbose = TRUE
)
```

Arguments

`data`	data.frame with the x- and y-coordinate of case locations and the date of the case.
`pos_vars`	vector of length two with the names of the columns containing the y and x coordinates, respectively.
`t_var`	character string with the name of the column with the date of the case. If single-period analysis then set t_var to NULL.
`format`	character string with the format of the date specified by t_var. See strptime
`verbose`	Logical indicating whether to print information

Details

Given a data frame containing the point location and date of cases, the function will return an sf object of the points with the date information.

Value

An sf object of the same size as `data`

Examples

```
dp <- data.frame(y=runif(10,0,3),x=runif(10,0,3),date=paste0("2021-01-",11:20))
dp <- create_points(dp,pos_vars = c('y','x'),t_var='date')
```

example_points *Simulated point data for running single-period examples*

Description

A set of 261 points simulated within the boundary of the city Birmingham, UK from a log-Gaussian Cox process.

Usage

```
example_points
```

Format

An object of class `data.frame` with 261 rows and 3 columns.

family.grid *Extracts the family from a* `grid` *object.*

Description

Extracts the family from a `grid` object.

Usage

```
## S3 method for class 'grid'
family(object, ...)
```

Arguments

`object`	A `grid` object.
`...`	Further arguments passed from other methods

Value

A family object.

family.rtsFit *Extracts the family from a* rtsFit *object.*

Description

Extracts the family from a rtsFit object.

Usage

```
## S3 method for class 'rtsFit'
family(object, ...)
```

Arguments

object	A rtsFit object.
...	Further arguments passed from other methods

Value

A family object.

fitted.rtsFit *Fitted values from a* rtsFit *object*

Description

Fitted values should not be generated directly from an rtsFit object, rather fitted values should be generated using the original grid object. A message is printed to the user.

Usage

```
## S3 method for class 'rtsFit'
fitted(object, ...)
```

Arguments

object	A rtsFit object.
...	Further arguments passed from other methods

Value

Nothing, called for effects.

fixed.effects *Extracts the fixed effect estimates*

Description

Extracts the fixed effect estimates from an `rtsFit object` returned from call of `lgcp_ml()` or `lgcp_bayes()` in the grid class.

Usage

```
fixed.effects(object)
```

Arguments

`object`	An `mcml` model fit.

Value

A named, numeric vector of fixed-effects estimates.

formula.grid *Extracts the formula from a* `grid` *object.*

Description

Extracts the formula from a `rtsFit` object stored in a grid object. Only returns the top level formula. For region models this is the formula at the region level, otherwise the grid-level formula is returned. No random effects specifications are included in the returned formula.

Usage

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

Arguments

`x`	A `grid` object.
`...`	Further arguments passed from other methods

Value

A formula object.

formula.rtsFit *Extracts the formula from a* rtsFit *object.*

Description

Extracts the formula from a rtsFit object. Only returns the top level formula. For region models this is the formula at the region level, otherwise the grid-level formula is returned. No random effects specifications are included in the returned formula.

Usage

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

Arguments

x	A rtsFit object.
...	Further arguments passed from other methods

Value

A formula object.

grid *An rts grid object*

Description

An rts grid object

An rts grid object

Details

An rts grid object is an R6 class holding the spatial data with data, model fitting, and analysis functions.

INTRODUCTION

The various methods of the class include examples and details of their implementation. The **sf** package is used for all spatial data. A typical workflow with this class would be:

1. Create a new grid object. The class is initialized with either a single polygon describing the area of interest or a collection of polygons if spatially aggregated data are used.

2. If the location (and times) of cases are available (i.e. the data are not spatially aggregated), then we map the points to the computational grid. The function create_points can generate point data in the correct **sf** format. The member function `points_to_grid` will then map these data to the grid. Counts can also be manually added to grid data. For region data, since the counts are assumed to be already aggregated, these must be manually provided by the user. The case counts must appear in columns with specific names. If there is only a single time period then the counts must be in a column named `y`. If there are multiple time periods then the counts must be in columns names `t1`, `t2`, `t3`,... Associated columns labelled `date1`, `date2`, etc. will permit use of some functionality regarding specific time intervals.
3. If any covariates are to be used for the modelling, then these can be mapped to the compuational grid using the function `add_covariates()`. Other functions, `add_time_indicators()` and `get_dow()` will also generate relevant temporal indicators where required. At a minimum we would recommend including a measure of population density.
4. Fit a model. There are multiple methods for model fitting, which are available through the member functions `lgcp_ml()` and `lgcp_bayes()` for maximum likelihood and Bayesian approaches, respectively. The results are stored internally and optionally returned as a `rtsFit` object.
5. Summarise the output. The main functions for summarising the output are `extract_preds()`, which will generate predictions of relative risk, incidence rate ratios, and predicted incidence, and `hotspots()`, which will estimate probabilities that these statistics exceed given thresholds. For spatially-aggregated data models, the relative risk applies to the grid, whereas rate ratios and predicted incidence applies to the areas.
6. Predictions can be visualised or aggregated to relevant geographies with the `plot()` and `aggregate()` functions.

Specific details of the implementation of each of these functions along with examples appear below.

PLOTTING

If `zcol` is not specified then only the geometry is plotted, otherwise the covariates specified will be plotted. The user can also use sf plotting functions on self$grid_data and self$region_data directly.

POINTS TO GRID

Given the sf object with the point locations and date output from `create_points()`, the functions will add columns to `grid_data` indicating the case count in each cell in each time period.

Case counts are generated for each grid cell for each time period. The user can specify the length of each time period; currently `day`, `week`, and `month` are supported.

The user must also specify the number of time periods to include with the `laglength` argument. The total number of time periods is the specified lag length counting back from the most recent case. The columns in the output will be named `t1`, `t2`,... up to the lag length, where the highest number is the most recent period.

ADDING COVARIATES

Spatially-varying data only

`cov_data` is an object describing covariate over the area of interest. sf, RasterLayer and SpatRaster objects are supported, with rasters converted internally to sf. The values are mapped onto `grid_data`. For each grid cell in `grid_data` a weighted average of each covariate listed in `zcols` is generated with weights either equal to the area of intersection of the grid cell and the polygons in `cov_data` (`weight_type="area"`), or this area multiplied by the population density of the polygon

for population weighted (`weight_type="pop"`). Columns with the names in `zcols` are added to the output.

Temporally-varying only data

`cov_data` is a data frame with number of rows equal to the number of time periods. One of the columns must be called `t` and have values from 1 to the number of time periods. The other columns of the data frame have the values of the covariates for each time period. See `get_dow()` for day of week data. A total of length(zcols)*(number of time periods) columns are added to the output: for each covariate there will be columns appended with each time period number. For example, `dayMon1`, `dayMon2`, etc.

Spatially and temporally varying data

There are two ways to add data that vary both spatially and temporally. The final output for use in analysis must have a column for each covariate and each time period with the same name appended by the time period number, e.g. `covariateA1`,`covariateA2`,... If the covariate values for different time periods are in separate sf objects, one can follow the method for spatially-varying only data above and append the time period number using the argument `t_label`. If the values for different time periods are in the same sf object then they should be named as described above and then can be added as for spatially-varying covariates, e.g. `zcols=c("covariateA1","covariateA2")`.

BAYESIAN MODEL FITTING

The grid data must contain columns `t*`, giving the case count in each time period (see `points_to_grid`), as well as any covariates to include in the model (see `add_covariates`) and the population density. Otherwise, if the data are regional data, then the outcome counts must be in self$region_data

Our statistical model is a Log Gaussian cox process, whose realisation is observed on the Cartesian area of interest A and time period T. The resulting data are relaisations of an inhomogeneous Poisson process with stochastic intensity function $\{\lambda s, t : s \in A, t \in T\}$. We specify a log-linear model for the intensity:

$$\lambda(s,t) = r(s,t)exp(X(s,t)'\gamma + Z(s,t))$$

where r(s,t) is a spatio-temporally varying Poisson offset. X(s,t) is a length Q vector of covariates including an intercept and Z(s,t) is a latent field. We use an auto-regressive specification for the latent field, with spatial innovation in each field specified as a spatial Gaussian process.

The argument `approx` specifies whether to use a full LGCP model (`approx='none'`) or whether to use either a nearest neighbour approximation (`approx='nngp'`) or a "Hilbert space" approximation (`approx='hsgp'`). For full details of NNGPs see XX and for Hilbert space approximations see references (1) and (2).

Priors

For Bayesian model fitting, the priors should be provided as a list to the griddata object:

```
griddata$priors <- list(
  prior_lscale=c(0,0.5),
  prior_var=c(0,0.5),
  prior_linpred_mean=c(-5,rep(0,7)),
  prior_linpred_sd=c(3,rep(1,7))
)
```

where these refer to the priors: `prior_lscale`: the length scale parameter has a half-normal prior $N(a, b^2)I[0, \infty)$. The vector is `c(a,b)`. `prior_var`: the standard deviation term has a half normal prior σ $N(a, b^2)I[0, \infty)$. The vector is `c(a,b)`. `prior_linpred_mean` and `prior_linpred_sd`: The parameters of the linear predictor. If X is the nT x Q matrix of covariates, with the first column as ones for the intercept, then the linear prediction contains the term $X'\gamma$. Each parameter in γ has prior γ_q $N(a_q, b_q^2)$. `prior_linpred_mean` should be the vector `(a_1,a_2,...,a_Q)` and `prior_linpred_sd` should be `(b_1,b_2,...,b_Q)`.

MAXIMUM LIKELIHOOD MODEL FITTING

The grid data must contain columns `t*`, giving the case count in each time period (see `points_to_grid`), as well as any covariates to include in the model (see `add_covariates`) and the population density. Otherwise, if the data are regional data, then the outcome counts must be in self$region_data. See `lgcp_bayes()` for more details on the model.

The argument `approx` specifies whether to use a full LGCP model (`approx='none'`) or whether to use either a nearest neighbour approximation (`approx='nngp'`)

Model fitting uses one of several stochastic maximum likelihood algorithms, which have three steps:

1. Sample random effects using MCMC. Using cmdstanr is recommended as it is much faster. The arguments `mcmc_warmup` and `mcmc_sampling` specify the warmup and sampling iterations for this step.
2. Fit fixed effect parameters using expectation maximisation.
3. Fit covariance parameters using expectation maximisation. This third step is the slowest. The NNGP approximation provides some speed improvements. Otherwise this step can be skipped if the covaraince parameters are "known". The argument `algo` specifies the algorithm, the user can select either MCMC maximum likelihood or stochastic approximation expectation maximisation with or without Ruppert-Polyak averaging. MCMC-ML can be used with or without adaptive MCMC sample sizes and either a derivative free or quasi-Newton optimiser (depending on the underlying model).

EXTRACTING PREDICTIONS

Three outputs can be extracted from the model fit, which will be added as columns to `grid_data`:

Predicted incidence: If type includes `pred` then `pred_mean_total` and `pred_mean_total_sd` provide the predicted mean total incidence and its standard deviation, respectively. `pred_mean_pp` and `pred_mean_pp_sd` provide the predicted population standardised incidence and its standard deviation.

Relative risk: if type includes `rr` then the relative risk is reported in the columns `rr` and `rr_sd`. The relative risk here is the exponential of the latent field, which describes the relative difference between expexted mean and predicted mean incidence.

Incidence risk ratio: if type includes `irr` then the incidence rate ratio (IRR) is reported in the columns `irr` and `irr_sd`. This is the ratio of the predicted incidence in the last period (minus `t_lag`) to the predicted incidence in the last period minus `irr_lag` (minus `t_lag`). For example, if the time period is in days then setting `irr_lag` to 7 and leaving `t_lag=0` then the IRR is the relative change in incidence in the present period compared to a week prior.

Public fields

`grid_data` sf object specifying the computational grid for the analysis

`region_data` sf object specifying an irregular lattice, such as census areas, within which case counts are aggregated. Only used if polygon data are provided on class initialisation.

`priors` list of prior distributions for the analysis

`bobyqa_control` list of control parameters for the BOBYQA algorithm, must contain named elements any or all of `npt`, `rhobeg`, `rhoend`, `covrhobeg`, `covrhoend`. Only has an effect for the HSGP and NNGP approximations. The latter two parameters control the covariance parameter optimisation, while the former control the linear predictor.

`boundary` sf object showing the boundary of the area of interest

Methods

Public methods:

- `grid$new()`
- `grid$print()`
- `grid$plot()`
- `grid$points_to_grid()`
- `grid$add_covariates()`
- `grid$get_dow()`
- `grid$add_time_indicators()`
- `grid$lgcp_bayes()`
- `grid$lgcp_ml()`
- `grid$extract_preds()`
- `grid$hotspots()`
- `grid$aggregate_output()`
- `grid$scale_conversion_factor()`
- `grid$get_region_data()`
- `grid$variogram()`
- `grid$reorder()`
- `grid$data()`
- `grid$get_random_effects()`
- `grid$model_fit()`
- `grid$clone()`

Method `new()`**:** Create a new grid object

Produces a regular grid over an area of interest as an sf object, see details for information on initialisation.

Usage:

```
grid$new(poly, cellsize, verbose = TRUE)
```

Arguments:

`poly` An sf object containing either one polygon describing the area of interest or multiple polygons representing survey or census regions in which the case data counts are aggregated

`cellsize` The dimension of the grid cells

`verbose` Logical indicating whether to provide feedback to the console.

Returns: NULL

Examples:

```
# a simple example with a square and a small number of cells
# this same running example is used for the other functions
b1 = sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)

# an example with multiple polygons
data("birmingham_crime")
g2 <- grid$new(birmingham_crime,cellsize = 1000)
```

Method `print()`**:** Prints this object

Usage:

```
grid$print()
```

Returns: None. called for effects.

Method `plot()`**:** Plots the grid data

Usage:

```
grid$plot(zcol)
```

Arguments:

`zcol` Vector of strings specifying names of columns of `grid_data` to plot

Returns: A plot

Examples:

```
b1 = sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
g1$plot()

# a plot with covariates - we simulate covariates first
g1$grid_data$cov <- stats::rnorm(nrow(g1$grid_data))
g1$plot("cov")
```

Method `points_to_grid()`**:** Generates case counts of points over the grid

Counts the number of cases in each time period in each grid cell

Usage:

```
grid$points_to_grid(
  point_data,
  t_win = c("day"),
  laglength = 14,
  date_format = "ymd",
  verbose = TRUE
)
```

Arguments:

`point_data` sf object describing the point location of cases with a column `t` of the date of the case in YYYY-MM-DD format. See create_points

`t_win` character string. One of "day", "week", or "month" indicating the length of the time windows in which to count cases

`laglength` integer The number of time periods to include counting back from the most recent time period

`date_format` String describing the format of the date in the data as a combination of "d" days, "m" months, and "y" years, either "dmy", "myd", "ymd", "ydm", "dym" "mdy" as used by the lubridate package.

`verbose` Logical indicating whether to report detailed output

Returns: NULL

Examples:

```
b1 <- sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
# simulate some points
dp <- data.frame(y=runif(10,0,3),x=runif(10,0,3),date=paste0("2021-01-",11:20))
dp <- create_points(dp,pos_vars = c('y','x'),t_var='date')
g1$points_to_grid(dp, laglength=5)
```

Method `add_covariates()`**:** Adds covariate data to the grid

Maps spatial, temporal, or spatio-temporal covariate data onto the grid.

Usage:

```
grid$add_covariates(
  cov_data,
  zcols,
  weight_type = "area",
  popdens = NULL,
  verbose = TRUE,
  t_label = NULL
)
```

Arguments:

`cov_data` sf, RasterLayer, SpatRaster object or a data.frame. See details.

`zcols` vector of character strings with the names of the columns of `cov_data` to include

`weight_type` character string. Either "area" for area-weighted average or "pop" for population-weighted average

`popdens` character string. The name of the column in `cov_data` with the population density. Required if weight_type="pop"

`verbose` logical. Whether to provide a progress bar

`t_label` integer. If adding spatio-temporally varying data by time period, this time label should be appended to the column name. See details.

Returns: NULL

Examples:

```
b1 <- sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
cov1 <- grid$new(b1,0.8)
cov1$grid_data$cov <- runif(nrow(cov1$grid_data))
```

```
g1$add_covariates(cov1$grid_data,
                  zcols="cov",
                  verbose = FALSE)

\donttest{
# mapping population data from some other polygons
data("boundary")
data("birmingham_crime")
g2 <- grid$new(boundary,cellsize=0.008)
msoa <- sf::st_transform(birmingham_crime,crs = 4326)
suppressWarnings(sf::st_crs(msoa) <- sf::st_crs(g2$grid_data)) # ensure crs matches
g2$add_covariates(msoa,
                  zcols="pop",
                  weight_type="area",
                  verbose=FALSE)
g2$plot("pop")
}
```

Method `get_dow()`**:** Generate day of week data

Create data frame with day of week indicators

Generates a data frame with indicator variables for each day of the week for use in the `add_covariates()` function.

Usage:

```
grid$get_dow()
```

Returns: data.frame with columns `t`, `day`, and `dayMon` to `daySun`

Examples:

```
b1 <- sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
dp <- data.frame(y=runif(10,0,3),x=runif(10,0,3),date=paste0("2021-01-",11:20))
dp <- create_points(dp,pos_vars = c('y','x'),t_var='date')
g1$points_to_grid(dp, laglength=5)
dow <- g1$get_dow()
g1$add_covariates(dow,zcols = colnames(dow)[3:ncol(dow)])
```

Method `add_time_indicators()`**:** Adds time period indicators to the data

Adds indicator variables for each time period to the data. To include these in a model fitting procedure use, for example, `covs = c("time1i, time2i,...)`

Usage:

```
grid$add_time_indicators()
```

Returns: Nothing. Called for effects.

Method `lgcp_bayes()`**:** Fit an (approximate) log-Gaussian Cox Process model using Bayesian methods

Usage:

```
grid$lgcp_bayes(
  popdens = NULL,
  covs = NULL,
  covs_grid = NULL,
  approx = "nngp",
  m = 10,
  L = 1.5,
  model = "exp",
  known_theta = NULL,
  iter_warmup = 500,
  iter_sampling = 500,
  chains = 3,
  parallel_chains = 3,
  verbose = TRUE,
  vb = FALSE,
  use_cmdstanr = FALSE,
  return_stan_fit = FALSE,
  ...
)
```

Arguments:

`popdens` character string. Name of the population density column

`covs` vector of character string. Base names of the covariates to include. For temporally-varying covariates only the stem is required and not the individual column names for each time period (e.g. `dayMon` and not `dayMon1`, `dayMon2`, etc.)

`covs_grid` If using a region model, covariates at the level of the grid can also be specified by providing their names to this argument.

`approx` Either "rank" for reduced rank approximation, or "nngp" for nearest neighbour Gaussian process.

`m` integer. Number of basis functions for reduced rank approximation, or number of nearest neighbours for nearest neighbour Gaussian process. See Details.

`L` integer. For reduced rank approximation, boundary condition as proportionate extension of area, e.g. `L=2` is a doubling of the analysis area. See Details.

`model` Either "exp" for exponential covariance function or "sqexp" for squared exponential covariance function

`known_theta` An optional vector of two values of the covariance parameters. If these are provided then the covariance parameters are assumed to be known and will not be estimated.

`iter_warmup` integer. Number of warmup iterations

`iter_sampling` integer. Number of sampling iterations

`chains` integer. Number of chains

`parallel_chains` integer. Number of parallel chains

`verbose` logical. Provide feedback on progress

`vb` Logical indicating whether to use variational Bayes (TRUE) or full MCMC sampling (FALSE)

`use_cmdstanr` logical. Defaults to false. If true then cmdstanr will be used instead of rstan.

`return_stan_fit` logical. The results of the model fit are stored internally as an `rstFit` object and returned in that format. If this argument is set to TRUE, then the fitted stan object will instead be returned, but the `rtsFit` object will still be saved.

... additional options to pass to '$sample()".

priors list. See Details

Returns: A stanfit or a CmdStanMCMC object

Examples:

```
# the data are just random simulated points
b1 <- sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
dp <- data.frame(y=runif(10,0,3),x=runif(10,0,3),date=paste0("2021-01-",11:20))
dp <- create_points(dp,pos_vars = c('y','x'),t_var='date')
cov1 <- grid$new(b1,0.8)
cov1$grid_data$cov <- runif(nrow(cov1$grid_data))
g1$add_covariates(cov1$grid_data,
                  zcols="cov",
                  verbose = FALSE)
g1$points_to_grid(dp, laglength=5)
g1$priors <- list(
  prior_lscale=c(0,0.5),
  prior_var=c(0,0.5),
  prior_linpred_mean=c(0),
  prior_linpred_sd=c(5)
  )
\donttest{
g1$lgcp_bayes(popdens="cov", approx = "hsgp", parallel_chains = 0)
g1$model_fit()
# we can extract predictions
g1$extract_preds("rr")
g1$plot("rr")
g1$hotspots(rr.threshold = 2)

 # this example uses real aggregated data but will take a relatively long time to run
 data("birmingham_crime")
 example_data <- birmingham_crime[,c(1:8,21)]
 example_data$y <- birmingham_crime$t12
 g2 <- grid$new(example_data,cellsize=1000)
 g2$priors <- list(
  prior_lscale=c(0,0.5),
  prior_var=c(0,0.5),
  prior_linpred_mean=c(-3),
  prior_linpred_sd=c(5)
)
g2$lgcp_bayes(popdens="pop", approx = "hsgp", parallel_chains = 0)
g2$model_fit()
g2$extract_preds("rr")
g2$plot("rr")
g2$hotspots(rr.threshold = 2)
}
```

Method `lgcp_ml()`**:** Fit an (approximate) log-Gaussian Cox Process model using Maximum

Likelihood

Usage:

```
grid$lgcp_ml(
  popdens = NULL,
  covs = NULL,
  covs_grid = NULL,
  approx = "nngp",
  m = 10,
  L = 1.5,
  model = "exp",
  known_theta = NULL,
  starting_values = NULL,
  lower_bound = NULL,
  upper_bound = NULL,
  formula_1 = NULL,
  formula_2 = NULL,
  algo = 4,
  alpha = 0.7,
  conv_criterion = 1,
  tol = 0.01,
  max.iter = 30,
  iter_warmup = 100,
  iter_sampling = 250,
  trace = 1,
  use_cmdstanr = FALSE
)
```

Arguments:

`popdens` character string. Name of the population density column

`covs` vector of strings. Base names of the covariates to include. For temporally-varying covariates only the stem is required and not the individual column names for each time period (e.g. `dayMon` and not `dayMon1`, `dayMon2`, etc.) Alternatively, a formula can be passed to the `formula` arguments below.

`covs_grid` If using a region model, covariates at the level of the grid can also be specified by providing their names to this argument. Alternatively, a formula can be passed to the `formula` arguments below.

`approx` Either "rank" for reduced rank approximation, or "nngp" for nearest neighbour Gaussian process.

`m` integer. Number of basis functions for reduced rank approximation, or number of nearest neighbours for nearest neighbour Gaussian process. See Details.

`L` integer. For reduced rank approximation, boundary condition as proportionate extension of area, e.g. `L=2` is a doubling of the analysis area. See Details.

`model` Either "exp" for exponential covariance function or "sqexp" for squared exponential covariance function

`known_theta` An optional vector of two values of the covariance parameters. If these are provided then the covariance parameters are assumed to be known and will not be estimated.

`starting_values` An optional list providing starting values of the model parameters. The list can have named elements `gamma` for the linear predictor parameters, `theta` for the covari-

ance parameters, and ar for the auto-regressive parameter. If there are covariates for the grid in a region data model then their parameters are gamma_g. The list elements must be a vector of starting values. If this is not provided then the non-intercept linear predictor parameters are initialised randomly as N(0,0.1), the covariance parameters as Uniform(0,0.5) and the auto-regressive parameter to 0.1.

lower_bound Optional. Vector of lower bound values for the fixed effect parameters.

upper_bound Optional. Vector of upper bound values for the fixed effect parameters.

formula_1 Optional. Instead of providing a list of covariates above (to covs) a formula can be specified here. For a regional model, this argument specified the regional-level fixed effects model.

formula_2 Optional. Instead of providing a list of covariates above (to covs_grid) a formula can be specified here. For a regional model, this argument specified the grid-level fixed effects model.

algo integer. 1 = MCMC ML with L-BFGS for beta and non-approximate covariance parameters, 2 = MCMC ML with BOBYQA for both, 3 = MCMC ML with L-BFGS for beta, BOBYQA for covariance parameters, 4 = SAEM with BOBYQA for both, 5 = SAEM with RP averaging and BOBYQA for both (default), 6-8 = as 1-3 but with adaptive MCMC sample size that starts at 20 with a max of iter_sampling

alpha Optional. Value for alpha in the SAEM parameter.

conv_criterion Integer. The convergence criterion for the algorithm. 1 = No improvement in the overall log-likelihood with probability 0.95, 2 = No improvement in the log-likelihood for beta with probability 0.95, 3 = Difference between model parameters is less than tol between iterations.

tol Scalar indicating the upper bound for the maximum absolute difference between parameter estimates on sucessive iterations, after which the algorithm terminates.

max.iter Integer. The maximum number of iterations for the algorithm.

iter_warmup integer. Number of warmup iterations

iter_sampling integer. Number of sampling iterations

trace Integer. Level of detail of information printed to the console. 0 = none, 1 = some (default), 2 = most.

use_cmdstanr logical. Defaults to false. If true then cmdstanr will be used instead of rstan.

... additional options to pass to $sample()

Returns: Optionally, an rtsFit model fit object. This fit is stored internally and can be retrieved with model_fit()

Examples:

```
# a simple example with completely random points
b1 <- sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
dp <- data.frame(y=runif(10,0,3),x=runif(10,0,3),date=paste0("2021-01-",11:20))
dp <- create_points(dp,pos_vars = c('y','x'),t_var='date')
cov1 <- grid$new(b1,0.8)
cov1$grid_data$cov <- runif(nrow(cov1$grid_data))
g1$add_covariates(cov1$grid_data,
                  zcols="cov",
                  verbose = FALSE)
```

```
g1$points_to_grid(dp, laglength=5)
\donttest{
g1$lgcp_ml(popdens="cov",iter_warmup = 100, iter_sampling = 50)
g1$model_fit()
g1$extract_preds("rr")
g1$plot("rr")
g1$hotspots(rr.threshold = 2)

# this example uses real aggregated data but will take a relatively long time to run
 data("birmingham_crime")
 example_data <- birmingham_crime[,c(1:8,21)]
 example_data$y <- birmingham_crime$t12
 g2 <- grid$new(example_data,cellsize=1000)
 g2$lgcp_ml(popdens = "pop",iter_warmup = 100, iter_sampling = 50)
 g2$model_fit()
 g2$extract_preds("rr")
 g2$plot("rr")
 g2$hotspots(rr.threshold = 2)
}
```

Method `extract_preds()`**:** Extract predictions

Extract incidence and relative risk predictions. The predictions will be extracted from the last model fit. If no previous model fit then use either `lgcp_ml()` or `lgcp_bayes()`, or see `model_fit()` to update the stored model fit.

Usage:

```
grid$extract_preds(
  type = c("pred", "rr", "irr"),
  irr.lag = NULL,
  t.lag = 0,
  popdens = NULL,
  verbose = TRUE
)
```

Arguments:

`type` Vector of character strings. Any combination of "pred", "rr", and "irr", which are, posterior mean incidence (overall and population standardised), relative risk, and incidence rate ratio, respectively.

`irr.lag` integer. If "irr" is requested as `type` then the number of time periods lag previous the ratio is in comparison to

`t.lag` integer. Extract predictions for previous time periods.

`popdens` character string. Name of the column in `grid_data` with the population density data

`verbose` Logical indicating whether to print messages to the console

Returns: NULL

Examples:

```
# See examples for lgcp_bayes() and lgcp_ml()
```

Method `hotspots()`**:** Generate hotspot probabilities

Generate hotspot probabilities. The last model fit will be used to extract predictions. If no previous model fit then use either `lgcp_ml()` or `lgcp_bayes()`, or see `model_fit()` to update the stored model fit.

Given a definition of a hotspot in terms of threshold(s) for incidence, relative risk, and/or incidence rate ratio, returns the probabilities each area is a "hotspot". See Details of `extract_preds`. Columns will be added to `grid_data`. Note that for incidence threshold, the threshold should be specified as the per individual incidence.

Usage:

```
grid$hotspots(
  incidence.threshold = NULL,
  irr.threshold = NULL,
  irr.lag = 1,
  rr.threshold = NULL,
  t.lag = 0,
  popdens,
  col_label = NULL
)
```

Arguments:

`incidence.threshold` Numeric. Threshold of population standardised incidence above which an area is a hotspot

`irr.threshold` Numeric. Threshold of incidence rate ratio above which an area is a hotspot.

`irr.lag` integer. Lag of time period to calculate the incidence rate ratio. Only required if `irr.threshold` is not `NULL`.

`rr.threshold` numeric. Threshold of local relative risk above which an area is a hotspot

`t.lag` integer. Extract predictions for incidence or relative risk for previous time periods.

`popdens` character string. Name of variable in `grid_data` specifying the population density. Needed if `incidence.threshold` is not `NULL`

`col_label` character string. If not NULL then the name of the column for the hotspot probabilities.

Returns: None, called for effects. Columns are added to grid or region data.

Examples:

```
\dontrun{
# See examples for lgcp_bayes() and lgcp_ml()
}
```

Method `aggregate_output()`**:** Aggregate output

Aggregate `lgcp_fit` output to another geography

Usage:

```
grid$aggregate_output(
  new_geom,
  zcols,
  weight_type = "area",
  popdens = NULL,
  verbose = TRUE
)
```

Arguments:

`new_geom` sf object. A set of polygons covering the same area as `boundary`

`zcols` vector of character strings. Names of the variables in `grid_data` to map to the new geography

`weight_type` character string, either "area" or "pop" for area-weighted or population weighted averaging, respectively

`popdens` character string. If `weight_type` is equal to "pop" then the name of the column in `grid_data` with population density data

`verbose` logical. Whether to provide progress bar.

Returns: An `sf` object identical to `new_geom` with additional columns with the variables specified in `zcols`

Examples:

```
\donttest{
b1 <- sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
dp <- data.frame(y=runif(10,0,3),x=runif(10,0,3),date=paste0("2021-01-",11:20))
dp <- create_points(dp,pos_vars = c('y','x'),t_var='date')
cov1 <- grid$new(b1,0.8)
cov1$grid_data$cov <- runif(nrow(cov1$grid_data))
g1$add_covariates(cov1$grid_data,
                  zcols="cov",
                  verbose = FALSE)
g1$points_to_grid(dp, laglength=5)
g1$priors <- list(
  prior_lscale=c(0,0.5),
  prior_var=c(0,0.5),
  prior_linpred_mean=c(0),
  prior_linpred_sd=c(5)
  )
res <- g1$lgcp_bayes(popdens="cov", parallel_chains = 1)
g1$extract_preds(res,
                 type=c("pred","rr"),
                 popdens="cov")
new1 <- g1$aggregate_output(cov1$grid_data,
                            zcols="rr")
}
```

Method `scale_conversion_factor()`**:** Returns scale conversion factor

Coordinates are scaled to `[-1,1]` for LGCP models fit with HSGP. This function returns the scaling factor for this conversion.

Usage:

```
grid$scale_conversion_factor()
```

Returns: numeric

Examples:

```
b1 = sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
g1$scale_conversion_factor()
```

Method `get_region_data()`**:** Returns summary data of the region/grid intersections

Information on the intersection between the region areas and the computational grid including the number of cells intersecting each region (`n_cell`), the indexes of the cells intersecting each region in order (`cell_id`), and the proportion of each region's area covered by each intersecting grid cell (`q_weights`).

Usage:

```
grid$get_region_data()
```

Returns: A named list

Method `variogram()`**:** Plots the empirical semi-variogram

Usage:

```
grid$variogram(popdens, yvar, nbins = 20)
```

Arguments:

`popdens` String naming the variable in the data specifying the offset. If not provided then no offset is used.

`yvar` String naming the outcome variable to calculate the variogram for. Optional, if not provided then the outcome count data will be used.

`nbins` The number of bins in the empirical semivariogram

Returns: A ggplot plot is printed and optionally returned

Method `reorder()`**:** Re-orders the computational grid

The quality of the nearest neighbour approximation can depend on the ordering of the grid cells. This function reorders the grid cells. If this is a region data model, then the intersections are recomputed.

Usage:

```
grid$reorder(option = "y", verbose = TRUE)
```

Arguments:

`option` Either "y" for order of the y coordinate, "x" for order of the x coordinate, "minimax" in which the next observation in the order is the one which maximises the minimum distance to the previous observations, or "random" which randomly orders them.

`verbose` Logical indicating whether to print a progress bar (TRUE) or not (FALSE).

Returns: No return, used for effects.

Method `data()`**:** A list of prepared data

The class prepares data for use in the in-built estimation functions. The same data could be used for alternative models. This is a utility function to facilitate model fitting for custom models.

Usage:

```
grid$data(m, approx, popdens, covs, covs_grid)
```

Arguments:

`m` The number of nearest neighbours or basis functions.

`approx` Either "rank" for reduced rank approximation, or "nngp" for nearest neighbour Gaussian process.

`popdens` String naming the variable in the data specifying the offset. If not provided then no offset is used.

`covs` An optional vector of covariate names. For regional data models, this is specifically for the region-level covariates.

`covs_grid` An optional vector of covariate names for region data models, identifying the covariates at the grid level.

Returns: A named list of data items used in model fitting

Method `get_random_effects()`**:** Returns the random effects stored in the object (if any) after using ML fitting. It's main use is if a fitting procedure is stopped, the random effects can still be returned.

Usage:

```
grid$get_random_effects()
```

Returns: A matrix of random effects samples if a MCMCML model has been initialised, otherwise returns FALSE

Method `model_fit()`**:** Either returns the stored last model fit with either `lgcp_ml` or `lgcp_bayes`, or updates the saved model fit if an object is provided.

Usage:

```
grid$model_fit(fit = NULL)
```

Arguments:

`fit` Optional. A previous `rtsFit` object. If provided then the function updates the internally stored model fit.

Returns: Either a `rtsFit` object or nothing if no model has been previously fit, or if the fit is updated.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
grid$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

(1) Solin A, Särkkä S. Hilbert space methods for reduced-rank Gaussian process regression. Stat Comput. 2020;30:419–46. doi:10.1007/s11222-019-09886-w.

(2) Riutort-Mayol G, Bürkner P-C, Andersen MR, Solin A, Vehtari A. Practical Hilbert space approximate Bayesian Gaussian processes for probabilistic programming. Stat Comput. 2023;33:17. doi:10.1007/s11222-022-10167-2.

See Also

create_points

points_to_grid, add_covariates

points_to_grid, add_covariates

Examples

```
## ------------------------------------------------
## Method `grid$new`
## ------------------------------------------------

# a simple example with a square and a small number of cells
# this same running example is used for the other functions
b1 = sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)

# an example with multiple polygons
data("birmingham_crime")
g2 <- grid$new(birmingham_crime,cellsize = 1000)

## ------------------------------------------------
## Method `grid$plot`
## ------------------------------------------------

b1 = sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
g1$plot()

# a plot with covariates - we simulate covariates first
g1$grid_data$cov <- stats::rnorm(nrow(g1$grid_data))
g1$plot("cov")

## ------------------------------------------------
## Method `grid$points_to_grid`
## ------------------------------------------------

b1 <- sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
# simulate some points
dp <- data.frame(y=runif(10,0,3),x=runif(10,0,3),date=paste0("2021-01-",11:20))
dp <- create_points(dp,pos_vars = c('y','x'),t_var='date')
g1$points_to_grid(dp, laglength=5)

## ------------------------------------------------
## Method `grid$add_covariates`
## ------------------------------------------------

b1 <-  sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
cov1 <- grid$new(b1,0.8)
cov1$grid_data$cov <- runif(nrow(cov1$grid_data))
g1$add_covariates(cov1$grid_data,
                  zcols="cov",
                  verbose = FALSE)


# mapping population data from some other polygons
data("boundary")
```

```
data("birmingham_crime")
g2 <- grid$new(boundary,cellsize=0.008)
msoa <- sf::st_transform(birmingham_crime,crs = 4326)
suppressWarnings(sf::st_crs(msoa) <- sf::st_crs(g2$grid_data)) # ensure crs matches
g2$add_covariates(msoa,
                  zcols="pop",
                  weight_type="area",
                  verbose=FALSE)
g2$plot("pop")


## ------------------------------------------------
## Method `grid$get_dow`
## ------------------------------------------------

b1 <- sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
dp <- data.frame(y=runif(10,0,3),x=runif(10,0,3),date=paste0("2021-01-",11:20))
dp <- create_points(dp,pos_vars = c('y','x'),t_var='date')
g1$points_to_grid(dp, laglength=5)
dow <- g1$get_dow()
g1$add_covariates(dow,zcols = colnames(dow)[3:ncol(dow)])

## ------------------------------------------------
## Method `grid$lgcp_bayes`
## ------------------------------------------------

# the data are just random simulated points
b1 <- sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
dp <- data.frame(y=runif(10,0,3),x=runif(10,0,3),date=paste0("2021-01-",11:20))
dp <- create_points(dp,pos_vars = c('y','x'),t_var='date')
cov1 <- grid$new(b1,0.8)
cov1$grid_data$cov <- runif(nrow(cov1$grid_data))
g1$add_covariates(cov1$grid_data,
                  zcols="cov",
                  verbose = FALSE)
g1$points_to_grid(dp, laglength=5)
g1$priors <- list(
  prior_lscale=c(0,0.5),
  prior_var=c(0,0.5),
  prior_linpred_mean=c(0),
  prior_linpred_sd=c(5)
  )

g1$lgcp_bayes(popdens="cov", approx = "hsgp", parallel_chains = 0)
g1$model_fit()
# we can extract predictions
g1$extract_preds("rr")
g1$plot("rr")
g1$hotspots(rr.threshold = 2)

 # this example uses real aggregated data but will take a relatively long time to run
```

```
 data("birmingham_crime")
 example_data <- birmingham_crime[,c(1:8,21)]
 example_data$y <- birmingham_crime$t12
 g2 <- grid$new(example_data,cellsize=1000)
 g2$priors <- list(
  prior_lscale=c(0,0.5),
  prior_var=c(0,0.5),
  prior_linpred_mean=c(-3),
  prior_linpred_sd=c(5)
)
g2$lgcp_bayes(popdens="pop", approx = "hsgp", parallel_chains = 0)
g2$model_fit()
g2$extract_preds("rr")
g2$plot("rr")
g2$hotspots(rr.threshold = 2)


## ------------------------------------------------
## Method `grid$lgcp_ml`
## ------------------------------------------------

# a simple example with completely random points
b1 <- sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
dp <- data.frame(y=runif(10,0,3),x=runif(10,0,3),date=paste0("2021-01-",11:20))
dp <- create_points(dp,pos_vars = c('y','x'),t_var='date')
cov1 <- grid$new(b1,0.8)
cov1$grid_data$cov <- runif(nrow(cov1$grid_data))
g1$add_covariates(cov1$grid_data,
                  zcols="cov",
                  verbose = FALSE)
g1$points_to_grid(dp, laglength=5)

g1$lgcp_ml(popdens="cov",iter_warmup = 100, iter_sampling = 50)
g1$model_fit()
g1$extract_preds("rr")
g1$plot("rr")
g1$hotspots(rr.threshold = 2)

# this example uses real aggregated data but will take a relatively long time to run
 data("birmingham_crime")
 example_data <- birmingham_crime[,c(1:8,21)]
 example_data$y <- birmingham_crime$t12
 g2 <- grid$new(example_data,cellsize=1000)
 g2$lgcp_ml(popdens = "pop",iter_warmup = 100, iter_sampling = 50)
 g2$model_fit()
 g2$extract_preds("rr")
 g2$plot("rr")
 g2$hotspots(rr.threshold = 2)



## ------------------------------------------------
```

```
## Method `grid$extract_preds`
## ------------------------------------------------

# See examples for lgcp_bayes() and lgcp_ml()

## ------------------------------------------------
## Method `grid$hotspots`
## ------------------------------------------------

## Not run:
# See examples for lgcp_bayes() and lgcp_ml()

## End(Not run)

## ------------------------------------------------
## Method `grid$aggregate_output`
## ------------------------------------------------


b1 <- sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
dp <- data.frame(y=runif(10,0,3),x=runif(10,0,3),date=paste0("2021-01-",11:20))
dp <- create_points(dp,pos_vars = c('y','x'),t_var='date')
cov1 <- grid$new(b1,0.8)
cov1$grid_data$cov <- runif(nrow(cov1$grid_data))
g1$add_covariates(cov1$grid_data,
                  zcols="cov",
                  verbose = FALSE)
g1$points_to_grid(dp, laglength=5)
g1$priors <- list(
  prior_lscale=c(0,0.5),
  prior_var=c(0,0.5),
  prior_linpred_mean=c(0),
  prior_linpred_sd=c(5)
  )
res <- g1$lgcp_bayes(popdens="cov", parallel_chains = 1)
g1$extract_preds(res,
                 type=c("pred","rr"),
                 popdens="cov")
new1 <- g1$aggregate_output(cov1$grid_data,
                            zcols="rr")


## ------------------------------------------------
## Method `grid$scale_conversion_factor`
## ------------------------------------------------

b1 = sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
g1 <- grid$new(b1,0.5)
g1$scale_conversion_factor()
```

logLik.rtsFit *Extracts the log-likelihood from an rtsFit object*

Description

Extracts the final log-likelihood value from an rtsFit object. Only returns a value for maximum likelihood model fits, otherwise it produces an error.

Usage

```
## S3 method for class 'rtsFit'
logLik(object, ...)
```

Arguments

object	An rtsFit model fit.
...	Further arguments passed from other methods

Value

An object of class logLik for maximum likelihood model fits, otherwise it returns an error.

predict.grid *Extract predictions from a* grid *object*

Description

Extract incidence and relative risk predictions. The predictions will be extracted from the last model fit in the grid object. If no previous model fit then use either grid$lgcp_ml() or grid$lgcp_bayes(), or see grid$model_fit() to update the stored model fit.

Usage

```
## S3 method for class 'grid'
predict(
  object,
  type = c("pred", "rr", "irr"),
  irr.lag = NULL,
  t.lag = 0,
  popdens = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

object	A grid object.
type	Vector of character strings. Any combination of "pred", "rr", and "irr", which are, posterior mean incidence (overall and population standardised), relative risk, and incidence rate ratio, respectively.
irr.lag	integer. If "irr" is requested as type then the number of time periods lag previous the ratio is in comparison to
t.lag	integer. Extract predictions for previous time periods.
popdens	character string. Name of the column in grid_data with the population density data
verbose	Logical indicating whether to print messages to the console
...	Further arguments passed from other methods

Details

Three outputs can be extracted from the model fit:

Predicted incidence: If type includes pred then pred_mean_total and pred_mean_total_sd provide the predicted mean total incidence and its standard deviation, respectively. pred_mean_pp and pred_mean_pp_sd provide the predicted population standardised incidence and its standard deviation. These are added to the grid data or to the regional data for spatially-aggregated data.

Relative risk: if type includes rr then the relative risk is reported in the columns rr and rr_sd. The relative risk here is the exponential of the latent field, which describes the relative difference between expected mean and predicted mean incidence. These are added to the grid data.

Incidence risk ratio: if type includes irr then the incidence rate ratio (IRR) is reported in the columns irr and irr_sd. This is the ratio of the predicted incidence in the last period (minus t_lag) to the predicted incidence in the last period minus irr_lag (minus t_lag). For example, if the time period is in days then setting irr_lag to 7 and leaving t_lag=0 then the IRR is the relative change in incidence in the present period compared to a week prior. These are added to the grid data or to the regional data for spatially-aggregated data.

Value

An sf object in which the predictions are stored.

Examples

```
# See examples for grid$lgcp_bayes() and grid$lgcp_ml()
```

predict.rtsFit *Predict from a* rtsFit *object*

Description

Predictions cannot be generated directly from an rtsFit object, rather new predictions should be generated using the original grid object. A message is printed to the user.

Usage

```
## S3 method for class 'rtsFit'
predict(object, ...)
```

Arguments

object	A rtsFit object.
...	Further arguments passed from other methods

Value

Nothing. Called for effects.

print.rtsFit *Prints an rtsFit fit output*

Description

Print method for class "rtsFit"

Usage

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

Arguments

x	an object of class "rtsFit"
...	Further arguments passed from other methods

Details

print.rtsFit tries to replicate the output of other regression functions, such as lm and lmer reporting parameters, standard errors, and z- and p- statistics for maximum likelihood esitmates, or posterior means, standard deviations and credible intervals for Bayesian models.

Value

No return value, called for side effects.

print.rtsFitSummary *Prints an rtsFitSummary fit output*

Description

Print method for class "rtsFitSummary"

Usage

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

Arguments

x	an object of class "rtsFitSummary"
...	Further arguments passed from other methods

Details

print.rtsFitSummary prints the summary of an rtsFit, see summary.rtsFit

Value

No return value, called for side effects.

progress_bar *Generates a progress bar*

Description

Prints a progress bar

Usage

```
progress_bar(i, n, len = 30)
```

Arguments

i	integer. The current iteration.
n	integer. The total number of interations
len	integer. Length of the progress a number of characters

Value

A character string

Examples

```
progress_bar(10,100)
```

`random.effects` *Extracts the random effect estimates*

Description

Extracts the random effect estimates or samples from an `rtsFit object` returned from call of `lgcp_ml()` or `lgcp_bayes()` in the grid class.

Usage

```
random.effects(object)
```

Arguments

`object`	An `mcml` model fit.

Value

A matrix of dimension (number of fixed effects) x (number of MCMC samples). For Laplace approximation, the number of "samples" equals one.

`residuals.grid` *Residuals method for a* `grid` *object*

Description

Conditional raw or standardised residuals are returned for a stored `rtsFit` objects. If no prior model fit is stored, then an error is returned.

Usage

```
## S3 method for class 'grid'
residuals(object, type, ...)
```

Arguments

`object`	A `grid` object.
`type`	Either "standardized" or "raw"
`...`	Further arguments passed from other methods

Value

A matrix with number of columns corresponding to the number of MCMC samples.

residuals.rtsFit *Residuals method for a* rtsFit *object*

Description

Conditional raw or standardised residuals for rstFit objects. The residuals are limited to conditional raw or standardised residuals currently to avoid copying the often large amount of model data stored in the associated grid object.

Usage

```
## S3 method for class 'rtsFit'
residuals(object, type, ...)
```

Arguments

object	A rtsFit object.
type	Either "standardized" or "raw"
...	Further arguments passed from other methods

Value

A matrix with number of columns corresponding to the number of MCMC samples.

summary.grid *Summarizes a* grid *object*

Description

Summarizes grid object.

Usage

```
## S3 method for class 'grid'
summary(object, ...)
```

Arguments

object	A grid object.
...	Further arguments passed from other methods

Value

Nothing. Called for effects.

summary.rtsFit *Summary method for class "rtsFit"*

Description

Summary method for class "rtsFit"

Usage

```
## S3 method for class 'rtsFit'
summary(object, ...)
```

Arguments

object	an object of class "rtsFit" as a result of a call to lgcp_ml() or lgcp_bayes()
...	Further arguments passed from other methods

Details

The summary methods aims to replicate the output of other regression model fitting functions and reports central point estimates, relevant test statistics, and uncertainty intervals. In addition, the returned summary object will also include time period specific relative risk and incidence predictions.

Value

An rtsFitSummary object

vcov.grid *Calculate Variance-Covariance matrix for a maximum likelihood object stored in* grid

Description

Returns the variance-covariance matrix for a LGCP object fit using maximum likelihood. If no relevant model is stored then the function returns an error

Usage

```
## S3 method for class 'grid'
vcov(object, ...)
```

Arguments

object	A grid object.
...	Further arguments passed from other methods

Value

A variance-covariance matrix.

`vcov.rtsFit` *Extract the Variance-Covariance matrix for a* `rtsFit` *object*

Description

Returns the calculated variance-covariance matrix for a `rtsFit` object that was fit using maximum likelihood methods. Bayesian models will return an error.

Usage

```
## S3 method for class 'rtsFit'
vcov(object, ...)
```

Arguments

`object`	A `rtsFit` object.
`...`	Further arguments passed from other methods

Value

A variance-covariance matrix.

Index