Package 'rts2'

October 27, 2022

Title Real-Time Disease Surveillance

Version 0.4

Description Supports modelling real-time case data to facilitate the real-time surveillance of infectious disease. A simple grid class structure is provided to generate a computational grid over an area of interest with methods to map covariates between geographies. An approximate log-Gaussian Cox Process model is fit using 'rstan' or 'cmdstanr' and provides output and analysis as 'sf' objects for simple visualisation. 'cmdstanr' can be downloaded at <https://mc-stan.org/cmdstanr/>. 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 (2020) <arXiv:2004.11408> and the nearest neighbour Gaussian process described by Datta et al (2016) <doi:10.1080/01621459.2015.1044091>.

License CC BY-SA 4.0

Encoding UTF-8

RoxygenNote 7.2.1

Biarch true

Depends R (>= 3.4.0)

Imports methods, R6, Rcpp (>= 0.12.0), RcppParallel (>= 5.0.1), rstan (>= 2.18.1), rstantools (>= 2.1.1), sf (>= 1.0-5), lubridate

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 (>= 2.18.1), StanHeaders (>= 2.18.0)

SystemRequirements GNU make

URL http://www.sam-watson.xyz/vignette.html

NeedsCompilation yes

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

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

Repository CRAN

Date/Publication 2022-10-27 15:02:39 UTC

R topics documented:

`rts2-package` *The 'rts2' package.*

Description

A DESCRIPTION OF THE PACKAGE

References

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

---

`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,
  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')
```

---

genNN *Generate matrix of nearest neighbours*

Description

Generates a M by n matrix of nearest neighbours. For observations with index less than M, only i-1 nearest neighbours are generated.

Usage

```
genNN(x, M)
```

Arguments

x	Matrix with x and y coordinates of the observations
M	Number of nearest neighbours

Value

A M by n matrix of nearest neighbours. Note the indexing of the observations returned by this function starts at zero.

grid *R6 class holding sf grid data with data and analysis functions*

Description

R6 class holding sf grid data with data and analysis functions

R6 class holding sf grid data with data and analysis functions

Details

Grid data consists of the computational grid over the area of interest. Outcomes and covariates are projected onto the grid, which can then be sent to the LGCP model.

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 grid$grid_data directly.

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.

Spatially-varying data only `cov_data`` is an sf object describing covariate values for a set of polygons over` 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")`.

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.

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.

We use the fast and accurate approximation for fully Bayesian Gaussian Processes proposed by Solin and Särkkä (1), using basis function approximations based on approximation via Laplace eigenfunctions for stationary covariance functions. See references (1) and (2) for complete details. The approximation is a linear sum of `m` eigenfunctions with the boundary conditions in each dimension `[-L,L]`. Coordinates in each dimension are scaled to `[-1,1]`, so L represents the proportionate extension of the analysis area.

Priors 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 $\sigma\ N(a, b^2)I[0, \infty)$. 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 $\gamma_q\ N(a_q, b_q^2)$. `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)`.

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 with the grid data

`priors` list of prior distributions for the analysis

`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$lgcp_fit()
- grid$extract_preds()
- grid$hotspots()
- grid$aggregate_output()
- grid$scale_conversion_factor()
- grid$clone()

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

Produces a regular grid over an area of interest as an sf object

Given a contiguous boundary describing an area of interest, which is stored as an sf object of a regular grid within the limits of the boundary at `$grid_data`. The boundary is also stored in the object as `$boundary`

Usage:

```
grid$new(boundary, cellsize)
```

Arguments:

`boundary` An sf object containing one polygon describing the area of interest

`cellsize` The dimension of the grid cells

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)
```

Method `print()`**:** Prints the $grid_data sf object

Usage:

```
grid$print()
```

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()
```

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

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.

Usage:

```
grid$points_to_grid(
  point_data,
  t_win = c("day"),
  laglength = 14,
  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

`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$new(b1,0.5)
dp <- data.frame(y=runif(10,0,3),x=runif(10,0,3),date=paste0("2021-01-",11:20))
dp <- create_points(dp,pos_vars = c('y','x'),t_var='date')
g1$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 object or 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)
```

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)
g1$get_dow()
```

Method `lgcp_fit()`**:** Fit an approximate log-Gaussian Cox Process model

Fit an approximate log-Gaussian Cox Process model

Usage:

```
grid$lgcp_fit(
  popdens,
  covs = NULL,
  approx = "nngp",
  m = 10,
  L = 1.5,
  model = "exp",
  dir = NULL,
  iter_warmup = 500,
  iter_sampling = 500,
```

```
  chains = 3,
  parallel_chains = 3,
  verbose = TRUE,
  use_cmdstanr = 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.)

`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

`dir` character string. Directory to save ouptut.

`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

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

`...` additional options to pass to '$sample()", see sample

`priors` list. See Details

Returns: A stanfit or a CmdStanMCMC object

Examples:

```
\dontrun{
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,
                  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_fit(popdens="cov")
}
```

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

Extract incidence and relative risk predictions

Usage:

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

Arguments:

`stan_fit` A stanfit or CmdStanMCMC object. Output of `lgcp_fit()`

`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

Returns: NULL

Examples:

```
\dontrun{
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,
                  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$lgcp_fit(popdens="cov")
g1$extract_preds(res,
                 type=c("pred","rr"),
```

```
                  popdens="cov")
}
```

Method `hotspots()`**:** Hotspots

Generate hotspot probabilities

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`

Usage:

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

Arguments:

`stan_fit` A stanfit or CmdStanMCMC object. Output of `lgcp_fit()`

`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

`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: NULL

Examples:

```
\dontrun{
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,
                  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$lgcp_fit(popdens="cov")
g1$hotspots(res,
            incidence.threshold=1,
            popdens="cov")
}
```

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:

```
\dontrun{
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,
                  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$lgcp_fit(popdens="cov")
g1$extract_preds(res,
                 type=c("pred","rr"),
                 popdens="cov")
new1 <- g1$aggregate_output(cov1,
                            zcols="rr")
}
```

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

Coordinates are scaled to `[-1,1]` for `lgcp_fit()`. 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 `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. 2020. http://arxiv.org/abs/2004.11408.

See Also

create_points

points_to_grid, add_covariates

Examples

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

```
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)

## ------------------------------------------------
## 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()

## ------------------------------------------------
## 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$new(b1,0.5)
dp <- data.frame(y=runif(10,0,3),x=runif(10,0,3),date=paste0("2021-01-",11:20))
dp <- create_points(dp,pos_vars = c('y','x'),t_var='date')
g1$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)

## ------------------------------------------------
## 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)
g1$get_dow()

## ------------------------------------------------
## Method `grid$lgcp_fit`
## ------------------------------------------------

## Not run:
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$add_covariates(cov1,
                  zcols="cov",
                  verbose = FALSE)
g1$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_fit(popdens="cov")

## End(Not run)

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

## Not run:
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,
                  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$lgcp_fit(popdens="cov")
g1$extract_preds(res,
                 type=c("pred","rr"),
                 popdens="cov")

## End(Not run)

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

## Not run:
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$add_covariates(cov1,
                  zcols="cov",
                  verbose = FALSE)
g1$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_fit(popdens="cov")
g1$hotspots(res,
            incidence.threshold=1,
            popdens="cov")

## End(Not run)

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

## Not run:
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,
                  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$lgcp_fit(popdens="cov")
g1$extract_preds(res,
                 type=c("pred","rr"),
                 popdens="cov")
new1 <- g1$aggregate_output(cov1,
                            zcols="rr")

## End(Not run)

## ------------------------------------------------
## 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()
```

`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)
```

Index