Package 'surveillance'

October 30, 2014

Title Temporal and Spatio-Temporal Modeling and Monitoring of Epidemic Phenomena

Version 1.8-1

Date 2014-10-29

Author Michael Höhle [aut, cre, ths], Sebastian Meyer [aut],Michaela Paul [aut], Leonhard Held [ctb, ths],Thais Correa [ctb], Mathias Hofmann [ctb], Christian Lang [ctb],Juliane Manitz [ctb], Andrea Riebler [ctb], Daniel Sabanés Bové [ctb],Maëlle Salmon [ctb], Dirk Schumacher [ctb], Stefan Steiner [ctb],Mikko Virtanen [ctb], Valentin Wimmer [ctb], R Core Team [ctb]
(A few code segments are modified versions of code from base R)

Maintainer Michael Höhle <hoehle@math.su.se>

Depends R (>= 3.0.2), methods, grDevices, graphics, stats, utils,Rcpp, sp (>= 1.0-15), xtable, polyCub (>= 0.4-3)

Imports MASS, Matrix, spatstat (>= 1.36-0)

LinkingTo Rcpp

Suggests parallel, grid, gridExtra, lattice, colorspace, scales,animation, msm, spc, quadprog, memoise, polyclip, rgeos,gpclib, maptools, intervals, spdep, numDeriv, maxLik, testthat,coda, splancs, gamlss, INLA, runjags

Description A package implementing statistical methods for the modeling and change-point detection in time series of counts, proportions and categorical data, as well as for the modeling of continuous-time epidemic phenomena, e.g. discrete-space setups such as the spatially enriched Susceptible-Exposed-Infectious-Recovered (SEIR) models for surveillance data, or continuous-space point process data such as the occurrence of disease or earthquakes. Main focus is on outbreak detection in count data time series originating from public health surveillance of infectious diseases, but applications could just as well originate from environmetrics, reliability engineering, econometrics or social sciences. Currently the package contains implementations of typical outbreak detection procedures such as Farrington et al (1996),Noufaily et al (2012) or the negative binomial LR-CUSUM method described in Hoehle and Paul (2008). Furthermore, inference

methods for the retrospective infectious disease model in Held et al (2005), Held et al (2006), Paul et al (2008) and Paul and Held (2011) are provided. A novel CUSUM approach combining logistic and multinomial logistic modelling is also included. Continuous self-exciting spatio-temporal point processes are modeled through additive-multiplicative conditional intensities as described in Höhle (2009) (``twinSIR'', discrete space) and Meyer et al (2012) (``twinstim'', continuous space). The package contains several real-world data sets, the ability to simulate outbreak data, visualize the results of the monitoring in temporal, spatial or spatio-temporal fashion. Note: The suggested package INLA is unfortunately not available from any mainstream repository - in case one wants to use the 'boda' algorithm one needs to manually install the INLA package as specified at http://www.r-inla.org/download.

License GPL-2

URL http://surveillance.r-forge.r-project.org/

BugReports https://r-forge.r-project.org/tracker/?group_id=45

Encoding latin1

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-10-30 15:54:15

R topics documented:

`surveillance-package` *Temporal and Spatio-Temporal Modeling and Monitoring of Epidemic Phenomena*

Description

A package implementing statistical methods for the modelling and change-point detection in time series of counts, proportions and categorical data, as well as for the modeling of continuous-time epidemic phenomena, e.g. discrete-space setups such as the spatially enriched Susceptible-Exposed-Infectious-Recovered (SEIR) models for surveillance data, or continuous-space point process data such as the occurrence of disease or earthquakes. Main focus is on outbreak detection in count data time series originating from public health surveillance of infectious diseases, but applications could just as well originate from environmetrics, reliability engineering, econometrics or social sciences. Currently the package contains implementations of typical outbreak detection procedures such as Stroup et. al (1989), Farrington et al (1996), Rossi et al (1999), Rogerson and Yamada (2001), a Bayesian approach, negative binomial CUSUM methods and a detector based on generalized likelihood ratios. Furthermore, inference methods for the retrospective infectious disease model in Held et al (2005), Held et al (2006), Paul et al (2008) and Paul and Held (2011) are provided. A novel CUSUM approach combining logistic and multinomial logistic modelling is also included. Continuous self-exciting spatio-temporal point processes are modeled through additive-multiplicative conditional intensities as described in Höhle (2009) ("twinSIR", discrete space) and Meyer et al (2012) ("twinstim", continuous space). The package contains several real-world datasets, the ability to simulate outbreak data, visualize the results of the monitoring in temporal, spatial or spatio-temporal fashion.

Details

Package:	surveillance
Version:	1.8-1
License:	GPL-2
URL:	http://surveillance.r-forge.r-project.org/

surveillance is an R package implementing statistical methods for the retrospective modeling and prospective change-point detection in time series of counts, proportions and categorical data. The main application is in the detection of aberrations in routine collected public health data seen as univariate and multivariate time series of counts or point-processes. However, applications could just as well originate from environmetrics, econometrics or social sciences. As many methods rely on statistical process control methodology, the package is thus also relevant to quality control and reliability engineering.

The fundamental data structure of the package is an S4 class `sts` wrapping observations, monitoring results and date handling for multivariate time series. Currently the package contains implementations typical outbreak detection procedures such as Stroup et al. (1989), Farrington et al., (1996),

Rossi et al. (1999), Rogerson and Yamada (2001), a Bayesian approach (Höhle, 2007), negative binomial CUSUM methods (Höhle and Mazick, 2009), and a detector based on generalized likelihood ratios (Höhle and Paul, 2008). However, also CUSUMs for the prospective change-point detection in binomial, beta-binomial and multinomial time series is covered based on generalized linear modelling. This includes e.g. paired binary CUSUM described by Steiner et al. (1999) or paired comparison Bradley-Terry modelling described in Höhle (2010). The package contains several real-world datasets, the ability to simulate outbreak data, visualize the results of the monitoring in temporal, spatial or spatio-temporal fashion.

Furthermore, the package contains inference methods for the retrospective infectious disease model in Held et al. (2005), Paul et al. (2008) ("algo.hhh") and Paul and Held (2011) ("hhh4") handling multivariate time series of counts. Furthermore, the fully Bayesian approach for univariate time series of counts from Held et al. (2006) ("twins") is also implemented. Self-exciting spatio-temporal point processes are modeled through additive-multiplicative conditional intensities as described in Höhle (2009) ("twinSIR") and Meyer et al (2012) ("twinstim").

Altogether, the package allows the modelling and monitoring of epidemic phenomena in temporal and spatio-temporal contexts.

Acknowledgements

Substantial contributions of code by:

Thais Correa, Leonhard Held, Mathias Hofmann, Christian Lang, Juliane Manitz, Andrea Riebler, Daniel Sabanés Bové, Maëlle Salmon, Dirk Schumacher, Stefan Steiner, Mikko Virtanen, Valentin Wimmer.

Furthermore, the authors would like to thank the following people for ideas, discussions, testing and feedback:

Doris Altmann, Johannes Dreesman, Johannes Elias, Marc Geilhufe, Kurt Hornik, Mayeul Kauffmann, Marcos Prates, Brian D. Ripley, Barry Rowlingson, Christopher W. Ryan, Klaus Stark, Yann Le Strat, André Michael Toschke, Wei Wei, Achim Zeileis.

Author(s)

Michael Höhle, Sebastian Meyer, Michaela Paul

Maintainer: Michael Höhle <hoehle@math.su.se>

References

See `citation(package="surveillance")`.

Examples

```
#Code from an early survey article about the package: Hoehle (2007)
#available from http://surveillance.r-forge.r-project.org/
## Not run: demo(cost)

#Code from a more recent book chapter about using the package for the
#monitoring of Danish mortality data (Hoehle and Mazick, 2010).
## Not run: demo(biosurvbook)
```

abattoir *Abattoir Data*

Description

A synthetic dataset from the Danish meat inspection – useful for illustrating the beta-binomial CUSUM.

Usage

```
data(abattoir)
```

Details

The object of class `sts` contains an artificial data set inspired by meat inspection data used by Danish Pig Production, Denmark. For each week the number of pigs with positive audit reports is recorded together with the total number of audits made that week.

References

Höhle, M. (2010), Changepoint detection in categorical time series, Book chapter to appear in T. Kneib and G. Tutz (Eds.), Statistical Modelling and Regression Structures, Springer.

See Also

`categoricalCUSUM`

Examples

```
data("abattoir")
plot(abattoir,legend.opts=NULL)
population(abattoir)
```

addSeason2formula *Function that adds a sine-/cosine formula to an existing formula.*

Description

This function helps to construct a `formula` object that can be used in a call to `hhh4` to model seasonal variation via a sum of sine and cosine terms.

Usage

```
addSeason2formula(f = ~1, S = 1, period = 52, timevar = "t")
```

Arguments

f	formula that the seasonal terms should be added to, defaults to an intercept ~1.
S	number of sine and cosine terms. If S is a vector, unit-specific seasonal terms are created.
period	period of the season, defaults to 52 for weekly data.
timevar	the time variable in the model. Defaults to "t".

Details

The function adds the seasonal terms

$$\sum_{s=1}^{S} \gamma_s \sin(\frac{2\pi s}{\texttt{period}} t) + \delta_s \cos(\frac{2\pi s}{\texttt{period}} t),$$

where γ_s and δ_s are the unknown parameters and t, $t = 1, 2, \ldots$ denotes the time variable `timevar`, to an existing formula `f`.

Note that the seasonal terms can also be expressed as

$$\gamma_s \sin(\frac{2\pi s}{\texttt{period}} t) + \delta_s \cos(\frac{2\pi s}{\texttt{period}} t) = A_s \sin(\frac{2\pi s}{\texttt{period}} t + \epsilon_s)$$

with amplitude $A_s = \sqrt{\gamma_s^2 + \delta_s^2}$ and phase difference $\tan(\epsilon_s) = \delta_s / \gamma_s$.

Value

Returns a `formula` object with the seasonal terms. Note that to use the resulting formula in `hhh4`, a time variable named as specified by the argument `timevar` must be available.

Author(s)

M. Paul, with contributions by S. Meyer

See Also

`hhh4`, `fe`, `ri`

Examples

```
# add 2 sine/cosine terms to a model with intercept and linear trend
addSeason2formula(f = ~ 1 + t, S = 2)

# the same for monthly data
addSeason2formula(f = ~ 1 + t, S = 2, period = 12)

# different number of seasons for a bivariate time series
addSeason2formula(f = ~ 1, S = c(3, 1), period = 52)
```

aggregate-methods *Aggregate the the series of an sts object*

Description

Method to aggregate the matrix slots of an sts object. Either the time series is aggregated so a new sampling frequency of `nfreq` units per time slot is obtained (i.e as in `aggregate.ts`) or the aggregation is over all `ncol` units.

Note: The function is not 100% consistent with what the generic function `aggregate` does.

Details

Warning: Aggregation by unit sets the upperbound slot to `NA` and the MAP object is left as-is, but the object cannot be plotted by unit any longer.

Methods

x = "sts", by="time", nfreq="all",... **x** an object of class `sts`

by a string being either "time" or "unit"

nfreq new sampling frequency if `by=="time"`. If `nfreq=="all"` then all time instances are summed.

... not used

returns an object of class `sts`

See Also

`aggregate`

Examples

```
data("ha")
has4 <- disProg2sts(ha)
dim(has4)
dim(aggregate(has4,by="unit"))
dim(aggregate(has4,nfreq=13))
```

aggregate.disProg *Aggregate the observed counts*

Description

Aggregates the observed counts for a multivariate `disProgObj` over the units. Future versions of `surveillance` will also allow for time aggregations etc.

Usage

```
  ## S3 method for class 'disProg'
aggregate(x,...)
```

Arguments

x	Object of class disProg
...	not used at the moment

Value

x	univariate disProg object with aggregated counts and respective states for each time point.

Examples

```
data(ha)
plot(aggregate(ha))
```

---

algo.bayes *The Bayes System*

Description

Evaluation of timepoints with the Bayes subsystem 1, 2, 3 or a self defined Bayes subsystem.

Usage

```
algo.bayesLatestTimepoint(disProgObj, timePoint = NULL,
     control = list(b = 0, w = 6, actY = TRUE,alpha=0.05))
algo.bayes(disProgObj, control = list(range = range,
     b = 0, w = 6, actY = TRUE,alpha=0.05))
algo.bayes1(disProgObj, control = list(range = range))
algo.bayes2(disProgObj, control = list(range = range))
algo.bayes3(disProgObj, control = list(range = range))
```

Arguments

disProgObj	object of class disProg (including the observed and the state chain)
timePoint	time point which shoud be evaluated in algo.bayes LatestTimepoint. The default is to use the latest timepoint
control	control object: range determines the desired timepoints which should be evaluated, b describes the number of years to go back for the reference values, w is the half window width for the reference values around the appropriate timepoint and actY is a boolean to decide if the year of timePoint also contributes w reference values. The parameter alpha is the $(1-\alpha)$-quantile to use in order to calculate the upper threshold. As default b, w, actY are set for the Bayes 1 system with alpha=0.05.

Details

Using the reference values the $(1-\alpha)\cdot 100\%$ quantile of the predictive posterior distribution is calculated as a threshold. An alarm is given if the actual value is bigger or equal than this threshold. It is possible to show using analytical computations that the predictive posterior in this case is the negative binomial distribution. Note: `algo.rki` or `algo.farrington` use two-sided prediction intervals – if one wants to compare with these procedures it is necessary to use an alpha, which is half the one used for these procedures.

Note also that `algo.bayes` calls `algo.bayesLatestTimepoint` for the values specified in `range` and for the system specified in `control`. `algo.bayes1`, `algo.bayes2`, `algo.bayes3` call `algo.bayesLatestTimepoint` for the values specified in `range` for the Bayes 1 system, Bayes 2 system or Bayes 3 system.

- `"Bayes 1"` reference values from 6 weeks. Alpha is fixed a t 0.05.
- `"Bayes 2"` reference values from 6 weeks ago and 13 weeks of the previous year (symmetrical around the same week as the current one in the previous year). Alpha is fixed at 0.05.
- `"Bayes 3"` 18 reference values. 9 from the year ago and 9 from two years ago (also symmetrical around the comparable week). Alpha is fixed at 0.05.

The procedure is now able to handle `NA`'s in the reference values. In the summation and when counting the number of observed reference values these are simply not counted.

Value

`survRes` `algo.bayesLatestTimepoint` returns a list of class `survRes` (surveillance result), which includes the alarm value for recognizing an outbreak (1 for alarm, 0 for no alarm), the threshold value for recognizing the alarm and the input object of class disProg. `algo.bayes` gives a list of class `survRes` which includes the vector of alarm values for every timepoint in `range` and the vector of threshold values for every timepoint in `range` for the system specified by `b`, `w` and `actY`, the range and the input object of class disProg. `algo.bayes1` returns the same for the Bayes 1 system, `algo.bayes2` for the Bayes 2 system and `algo.bayes3` for the Bayes 3 system.

Author(s)

M. Höhle, A. Riebler, C. Lang

Source

Riebler, A. (2004), Empirischer Vergleich von statistischen Methoden zur Ausbruchserkennung bei Surveillance Daten, Bachelor's thesis.

See Also

`algo.call`, `algo.rkiLatestTimepoint` and `algo.rki` for the RKI system.

Examples

```
disProg <- sim.pointSource(p = 0.99, r = 0.5, length = 208, A = 1,
                           alpha = 1, beta = 0, phi = 0,
                           frequency = 1, state = NULL, K = 1.7)

# Test for bayes 1 the latest timepoint
algo.bayesLatestTimepoint(disProg)

# Test week 200 to 208 for outbreaks with a selfdefined bayes
algo.bayes(disProg, control = list(range = 200:208, b = 1,
                                   w = 5, actY = TRUE,alpha=0.05))
# The same for bayes 1 to bayes 3
algo.bayes1(disProg, control = list(range = 200:208,alpha=0.05))
algo.bayes2(disProg, control = list(range = 200:208,alpha=0.05))
algo.bayes3(disProg, control = list(range = 200:208,alpha=0.05))
```

---

algo.call *Query Transmission to Specified Surveillance Algorithm*

Description

Transmission of a object of class disProg to the specified surveillance algorithm.

Usage

```
algo.call(disProgObj, control = list(
                  list(funcName = "rki1", range = range),
                  list(funcName = "rki", range = range,
                       b = 2, w = 4, actY = TRUE),
                  list(funcName = "rki", range = range,
                       b = 2, w = 5, actY = TRUE)))
```

Arguments

disProgObj	object of class disProg, which includes the state chain and the observed
control	specifies which surveillance algorithm should be used with their parameters. The parameter funcName and range must be specified. Here, funcName is the appropriate method function (without 'algo.') and range defines the time-points to be evaluated by the actual system. If control includes name this name is used in the survRes Object as name.

Value

list of survRes Objects
generated by the specified surveillance algorithm

See Also

algo.rki, algo.bayes, algo.farrington

Examples

```
# Create a test object
disProg <- sim.pointSource(p = 0.99, r = 0.5, length = 400, A = 1,
                           alpha = 1, beta = 0, phi = 0,
                           frequency = 1, state = NULL, K = 1.7)

# Let this object be tested from any methods in range = 200:400
range <- 200:400
survRes <- algo.call(disProg,
              control = list(
                        list(funcName = "rki1", range = range),
                        list(funcName = "rki2", range = range),
                        list(funcName = "rki3", range = range),
                        list(funcName = "rki", range = range,
                             b = 3, w = 2, actY = FALSE),
                        list(funcName = "rki", range = range,
                             b = 2, w = 9, actY = TRUE),
                        list(funcName = "bayes1", range = range),
                        list(funcName = "bayes2", range = range),
                        list(funcName = "bayes3", range = range),
                        list(funcName = "bayes", name = "myBayes",
                             range = range, b = 1, w = 5, actY = TRUE,alpha=0.05)
                         ) )
# this are some survResObjects
plot(survRes[["rki(6,6,0)"]])
survRes[["bayes(5,5,1)"]]
```

`algo.cdc` *The CDC Algorithm*

Description

Surveillance using the CDC Algorithm

Usage

```
algo.cdcLatestTimepoint(disProgObj, timePoint = NULL,
     control = list(b = 5, m = 1, alpha=0.025))
algo.cdc(disProgObj, control = list(range = range, b= 5, m=1,
     alpha = 0.025))
```

Arguments

`disProgObj`	object of class disProg (including the observed and the state chain).
`timePoint`	time point which shoud be evaluated in `algo.cdcLatestTimepoint`. The default is to use the latest timepoint.
`control`	control object: `range` determines the desired timepoints which should be evaluated, `b` describes the number of years to go back for the reference values, `m` is the half window width for the reference values around the appropriate timepoint (see details). The standard definition is `b`=5 and `m`=1.

Details

Using the reference values for calculating an upper limit, alarm is given if the actual value is bigger than a computed threshold. `algo.cdc` calls `algo.cdcLatestTimepoint` for the values specified in `range` and for the system specified in `control`. The threshold is calculated from the predictive distribution, i.e.

$$mean(x) + z_{\alpha/2} * sd(x) * \sqrt{(1 + 1/k)},$$

which corresponds to Equation 8-1 in Farrington and Andrews (2003).

Note that an aggregation into 4-week blocks occurs in `algo.cdcLatestTimepoint` and `m` denotes number of 4-week blocks (months) to use as reference values. This function currently does the same for monthly data (not correct!)

Value

`survRes` — `algo.cdcLatestTimepoint` returns a list of class `survRes` (surveillance result), which includes the alarm value (alarm = 1, no alarm = 0) for recognizing an outbreak, the threshold value for recognizing the alarm and the input object of class disProg.

`algo.cdc` gives a list of class `survRes` which includes the vector of alarm values for every timepoint in `range`, the vector of threshold values for every timepoint in `range` for the system specified by `b`, `w`, the range and the input object of class disProg.

Author(s)

M. Höhle

Source

Stroup, D., G. Williamson, J. Herndon, and J. Karon (1989). Detection of aberrations in the occurence of notifiable diseases surveillance data. Statistics in Medicine 8, 323-329.

Farrington, C. and N. Andrews (2003). Monitoring the Health of Populations, Chapter Outbreak Detection: Application to Infectious Disease Surveillance, pp. 203-231. Oxford University Press.

See Also

`algo.rkiLatestTimepoint`,`algo.bayesLatestTimepoint` and `algo.bayes` for the Bayes system.

Examples

```
# Create a test object
disProgObj <- sim.pointSource(p = 0.99, r = 0.5, length = 500,
                  A = 1,alpha = 1, beta = 0, phi = 0,
                  frequency = 1, state = NULL, K = 1.7)

# Test week 200 to 208 for outbreaks with a selfdefined cdc
algo.cdc(disProgObj, control = list(range = 400:500,alpha=0.025))
```

`algo.compare` *Comparison of Specified Surveillance Systems using Quality Values*

Description

Comparison of specified surveillance algorithms using quality values.

Usage

```
algo.compare(survResList)
```

Arguments

`survResList`	a list of survRes objects to compare via quality values.

Value

`matrix`	Matrix with values from `algo.quality`, i.e. quality values for every surveillance algorithm found in `survResults`.

See Also

`algo.quality`

Examples

```
# Create a test object
disProgObj <- sim.pointSource(p = 0.99, r = 0.5, length = 400,
                              A = 1, alpha = 1, beta = 0, phi = 0,
                              frequency = 1, state = NULL, K = 1.7)

# Let this object be tested from any methods in range = 200:400
range <- 200:400
survRes <- algo.call(disProgObj,
                     control = list(
                         list(funcName = "rki1", range = range),
                         list(funcName = "rki2", range = range),
                         list(funcName = "rki3", range = range),
                         list(funcName = "rki", range = range,
                                     b = 3, w = 2, actY = FALSE),
                         list(funcName = "rki", range = range,
                                     b = 2, w = 9, actY = TRUE),
                         list(funcName = "bayes1", range = range),
                         list(funcName = "bayes2", range = range),
                         list(funcName = "bayes3", range = range),
                         list(funcName = "bayes", name = "myBayes",
                         range = range, b = 1, w = 5, actY = TRUE,alpha=0.05)
                     ) )
algo.compare(survRes)
```

`algo.cusum` *CUSUM method*

Description

Approximate one-side CUSUM method for a Poisson variate based on the cumulative sum of the deviation between a reference value k and the transformed observed values. An alarm is raised if the cumulative sum equals or exceeds a prespecified decision boundary h. The function can handle time varying expectations.

Usage

```
algo.cusum(disProgObj, control = list(range = range, k = 1.04, h = 2.26,
           m = NULL, trans = "standard", alpha = NULL))
```

Arguments

`disProgObj` object of class disProg (including the observed and the state chain)

`control` control object:

`range` determines the desired time points which should be evaluated

`k` is the reference value

`h` the decision boundary

`m` how to determine the expected number of cases – the following arguments are possible

`numeric` a vector of values having the same length as `range`. If a single numeric value is specified then this value is replicated `length(range)` times.

`NULL` A single value is estimated by taking the mean of all observations previous to the first `range` value.

`"glm"` A GLM of the form

$$\log(m_t) = \alpha + \beta t + \sum_{s=1}^{S}(\gamma_s \sin(\omega_s t) + \delta_s \cos(\omega_s t)),$$

where $\omega_s = \frac{2\pi}{52}s$ are the Fourier frequencies is fitted. Then this model is used to predict the `range` values.

`trans` one of the following transformations (warning: anscombe and negbin transformations are experimental)

`rossi` standardized variables z3 as proposed by Rossi

`standard` standardized variables z1 (based on asympotic normality)

`anscombe` anscombe residuals – experimental

`anscombe2nd` anscombe residuals as in Pierce and Schafer (1986) based on 2nd order approximation of E(X) – experimental

`pearsonNegBin` compute Pearson residuals for NegBin – experimental

`anscombeNegBin` anscombe residuals for NegBin – experimental

`none` no transformation

`alpha` parameter of the negative binomial distribution, s.t. the variance is $m + \alpha * m^2$

Details

This implementation is experimental, but will not be developed further.

Value

`survRes` `algo.cusum` gives a list of class `survRes` which includes the vector of alarm values for every timepoint in `range` and the vector of cumulative sums for every timepoint in `range` for the system specified by `k` and `h`, the range and the input object of class disProg.

The `upperbound` entry shows for each time instance the number of diseased individuals it would have taken the cusum to signal. Once the CUSUM signals no resetting is applied, i.e. signals occurs until the CUSUM statistic again returns below the threshold.

The `control$m.glm` entry contains the fitted glm object, if the original argument was "`glm`".

Author(s)

M. Paul and M. Höhle

References

G. Rossi, L. Lampugnani and M. Marchi (1999), An approximate CUSUM procedure for surveillance of health events, Statistics in Medicine, 18, 2111–2122

D. A. Pierce and D. W. Schafer (1986), Residuals in Generalized Linear Models, Journal of the American Statistical Association, 81, 977–986

Examples

```
# Xi ~ Po(5), i=1,...,500
disProgObj <- create.disProg(week=1:500, observed= rpois(500,lambda=5),
                             state=rep(0,500))
# there should be no alarms as mean doesn't change
res <- algo.cusum(disProgObj, control = list(range = 100:500,trans="anscombe"))
plot(res)

# simulated data
disProgObj <- sim.pointSource(p = 1, r = 1, length = 250,
                          A = 0, alpha = log(5), beta = 0, phi = 10,
                          frequency = 10, state = NULL, K = 0)
plot(disProgObj)
```

```
# Test week 200 to 250 for outbreaks
surv <- algo.cusum(disProgObj, control = list(range = 200:250))
plot(surv)
```

`algo.farrington` *Surveillance for a count data time series using the Farrington method.*

Description

The function takes `range` values of the surveillance time series `disProgObj` and for each time point uses a GLM to predict the number of counts according to the procedure by Farrington et al. (1996). This is then compared to the observed number of counts. If the observation is above a specific quantile of the prediction interval, then an alarm is raised.

Usage

```
algo.farrington(disProgObj, control=list(range=NULL, b=3, w=3,
reweight=TRUE,verbose=FALSE,alpha=0.01,trend=TRUE,limit54=c(5,4),
powertrans="2/3",
fitFun=c("algo.farrington.fitGLM.fast","algo.farrington.fitGLM",
         "algo.farrington.fitGLM.populationOffset")))
```

Arguments

`disProgObj` object of class disProgObj (including the `observed` and the `state` time series.)

`control` Control object

- `range` Specifies the index of all timepoints which should be tested. If range is `NULL` the maximum number of possible weeks is used (i.e. as many weeks as possible while still having enough reference values).
- `b` how many years back in time to include when forming the base counts.
- `w` windows size, i.e. number of weeks to include before and after the current week
- `reweight` Boolean specifying whether to perform reweight step
- `trend` If `true` a trend is included and kept in case the conditions documented in Farrington et al. (1996) are met (see the results). If `false` then NO trend is fit.
- `verbose` Boolean indicating whether to show extra debugging information.
- `plot` Boolean specifying whether to show the final GLM model fit graphically (use History|Recording to see all pictures).
- `powertrans` Power transformation to apply to the data. Use either "2/3" for skewness correction (Default), "1/2" for variance stabilizing transformation or "none" for no transformation.
- `alpha` An approximate (two-sided) $(1-\alpha)$ prediction interval is calculated.

`limit54` To avoid alarms in cases where the time series only has about 0-2 cases the algorithm uses the following heuristic criterion (see Section 3.8 of the Farrington paper) to protect against low counts: no alarm is sounded if fewer than $cases = 5$ reports were received in the past $period = 4$ weeks. `limit54=c(cases,period)` is a vector allowing the user to change these numbers. Note: As of version 0.9-7 the term "last" period of weeks includes the current week - otherwise no alarm is sounded for horrible large numbers if the four weeks before that are too low.

`fitFun` String containing the name of the fit function to be used for fitting the GLM. The options are `algo.farrington.fitGLM.fast` (default) and `algo.farrington.fitGLM` or `algo.farrington.fitGLM.populationOffset`. See details of `algo.farrington.fitGLM` for more information.

Details

The following steps are perfomed according to the Farrington et al. (1996) paper.

1. fit of the initial model and initial estimation of mean and overdispersion.
2. calculation of the weights omega (correction for past outbreaks)
3. refitting of the model
4. revised estimation of overdispersion
5. rescaled model
6. omission of the trend, if it is not significant
7. repetition of the whole procedure
8. calculation of the threshold value
9. computation of exceedance score

Value

An object of class `SurvRes`.

Author(s)

M. Höhle

Source

A statistical algorithm for the early detection of outbreaks of infectious disease, Farrington, C.P., Andrews, N.J, Beale A.D. and Catchpole, M.A. (1996), J. R. Statist. Soc. A, 159, 547-563.

See Also

`algo.farrington.fitGLM`,`algo.farrington.threshold`

Examples

```
#Read Salmonella Agona data
data("salmonella.agona")

#Do surveillance for the last 100 weeks.
n <- length(salmonella.agona$observed)
#Set control parameters.
control <- list(b=4,w=3,range=(n-100):n,reweight=TRUE, verbose=FALSE,alpha=0.01)
res <- algo.farrington(salmonella.agona,control=control)
#Plot the result.
plot(res,disease="Salmonella Agona",method="Farrington")

## Not run:
#Generate random data and convert into sts object
set.seed(123)
x <- matrix(rpois(1000,lambda=1),ncol=1)
sts <- new("sts", observed=x, epoch=1:nrow(x), state=x*0, freq=52)

#Compare timing of the two possible fitters for algo.farrington (here using S4)
system.time( sts1 <- farrington(sts, control=list(range=c(500:1000),
                        fitFun="algo.farrington.fitGLM.fast")))
system.time( sts2 <- farrington(sts, control=list(range=c(500:1000),
                        fitFun="algo.farrington.fitGLM")))

#Check if results are the same
sum(upperbound(sts1) - upperbound(sts2))

## End(Not run)
```

---

algo.farrington.assign.weights

Assign weights to base counts

Description

Weights are assigned according to the Anscombe residuals

Usage

```
algo.farrington.assign.weights(s, weightsThreshold=1)
```

Arguments

s	Vector of standardized Anscombe residuals
weightsThreshold	A scalar indicating when observations are seen as outlier. In the original Farrington proposal the value was 1 (default value), in the improved version this value is suggested to be 2.58.

Value

Weights according to the residuals

See Also

See Also as `anscombe.residuals`

`algo.farrington.fitGLM` *Fit the Poisson GLM of the Farrington procedure for a single time point*

Description

The function fits a Poisson regression model (GLM) with mean predictor

$$\log \mu_t = \alpha + \beta t$$

as specified by the Farrington procedure. If requested, Anscombe residuals are computed based on an initial fit and a 2nd fit is made using weights, where base counts suspected to be caused by earlier outbreaks are downweighted.

Usage

```
algo.farrington.fitGLM(response, wtime, timeTrend = TRUE,
                       reweight = TRUE, ...)
algo.farrington.fitGLM.fast(response, wtime, timeTrend = TRUE,
                       reweight = TRUE, ...)
algo.farrington.fitGLM.populationOffset(response, wtime, population,
                       timeTrend=TRUE,reweight=TRUE, ...)
```

Arguments

`response`	The vector of observed base counts
`wtime`	Vector of week numbers corresponding to `response`
`timeTrend`	Boolean whether to fit the βt or not
`reweight`	Fit twice – 2nd time with Anscombe residuals
`population`	Population size. Possibly used as offset, i.e. in `algo.farrington.fitGLM.populationOffset` the value `log(population)` is used as offset in the linear predictor of the GLM: $$\log \mu_t = \log(\texttt{population}) + \alpha + \beta t$$ This provides a way to adjust the Farrington procedure to the case of greatly varying populations. Note: This is an experimental implementation with methodology not covered by the original paper.
`...`	Used to catch additional arguments, currently not used.

Details

Compute weights from an initial fit and rescale using Anscombe based residuals as described in the `anscombe.residuals` function.

Note that `algo.farrington.fitGLM` uses the `glm` routine for fitting. A faster alternative is provided by `algo.farrington.fitGLM.fast` which uses the `glm.fit` function directly (thanks to Mikko Virtanen). This saves computational overhead and increases speed for 500 monitored time points by a factor of approximately two. However, some of the routine `glm` functions might not work on the output of this function. Which function is used for `algo.farrington` can be controlled by the `control$fitFun` argument.

Value

An object of class GLM with additional fields `wtime`, `response` and `phi`. If the `glm` returns without convergence `NULL` is returned.

See Also

`anscombe.residuals`,`algo.farrington`

`algo.farrington.threshold`

Compute prediction interval for a new observation

Description

Depending on the current transformation $h(y) = \{y, \sqrt{y}, y^{2/3}\}$,

$$V(h(y_0) - h(\mu_0)) = V(h(y_0)) + V(h(\mu_0))$$

is used to compute a prediction interval. The prediction variance consists of a component due to the variance of having a single observation and a prediction variance.

Usage

```
algo.farrington.threshold(pred,phi,alpha=0.01,skewness.transform="none",y)
```

Arguments

`pred`	A GLM prediction object
`phi`	Current overdispersion parameter (superflous?)
`alpha`	Quantile level in Gaussian based CI, i.e. an $(1-\alpha)\cdot 100\%$ confidence interval is computed.
`skewness.transform`	Skewness correction, i.e. one of `"none"`, `"1/2"`, or `"2/3"`.
`y`	Observed number

Value

Vector of length four with lower and upper bounds of an $(1-\alpha)\cdot 100\%$ confidence interval (first two arguments) and corresponding quantile of observation y together with the median of the predictive distribution.

`algo.glrnb` *Count Data Regression Charts*

Description

Count data regression charts for the monitoring of surveillance time series.

Usage

```
algo.glrnb(disProgObj,control = list(range=range,c.ARL=5,
         mu0=NULL, alpha=0, Mtilde=1, M=-1, change="intercept",
         theta=NULL,dir=c("inc","dec"),ret=c("cases","value")))
```

Arguments

`disProgObj` object of class `disProg` to do surveillance for

`control` A list controlling the behaviour of the algorithm

`range` vector of indices in the observed vector to monitor (should be consecutive)

`mu0` A vector of in-control values of the mean of the negative binomial distribution with the same length as `range`. If `NULL` the observed values in `1:(min(range)-1)` are used to estimate beta through a generalized linear model. To fine-tune the model one can instead specify `mu0` as a list with two components:

`S` number of harmonics to include

`trend` include a term `t` in the GLM model

`alpha` The (known) dispersion parameter of the negative binomial distribution. If `alpha=0` then the negative binomial distribution boils down to the Poisson distribution and a call of `algo.glrnb` is equivalent to a call to `algo.glrpois`. If `alpha=NULL` the parameter is calculated as part of the in-control estimation.

`c.ARL` threshold in the GLR test, i.e. c_γ

`Mtilde` number of observations needed before we have a full rank the typical setup for the "`intercept`" and "`epi`" charts is `Mtilde=1`

`M` number of time instances back in time in the window-limited approach, i.e. the last value considered is $\max 1, n - M$. To always look back until the first observation use `M=-1`.

`change` a string specifying the type of the alternative. Currently the two choices are `intercept` and `epi`. See the SFB Discussion Paper 500 for details.

`theta` if `NULL` then the GLR scheme is used. If not `NULL` the prespecified value for κ or λ is used in a recursive LR scheme, which is faster.

`dir` a string specifying the direction of testing in GLR scheme. With `"inc"` only increases in x are considered in the GLR-statistic, with `"dec"` decreases are regarded.

`ret` a string specifying the type of `upperbound`-statistic that is returned. With `"cases"` the number of cases that would have been necessary to produce an alarm or with `"value"` the glr-statistic is computed (see below).

Details

This function implements the seasonal cound data chart based on generalized likelihood ratio (GLR) as described in the Hoehle and Paul (2008) paper. A moving-window generalized likelihood ratio detector is used, i.e. the detector has the form

$$N = \inf\left\{ n : \max_{1\leq k\leq n} \left[\sum_{t=k}^{n} \log\left\{ \frac{f_{\theta_1}(x_t|z_t)}{f_{\theta_0}(x_t|z_t)} \right\} \right] \geq c_\gamma \right\}$$

where instead of $1 \leq k \leq n$ the GLR statistic is computed for all $k \in \{n-M, \ldots, n-\tilde{M}+1\}$. To achieve the typical behaviour from $1 \leq k \leq n$ use `Mtilde=1` and `M=-1`.

So N is the time point where the GLR statistic is above the threshold the first time: An alarm is given and the surveillance is resetted starting from time $N+1$. Note that the same `c.ARL` as before is used, but if `mu0` is different at $N+1, N+2, \ldots$ compared to time $1, 2, \ldots$ the run length properties differ. Because `c.ARL` to obtain a specific ARL can only be obtained my Monte Carlo simulation there is no good way to update `c.ARL` automatically at the moment. Also, FIR GLR-detectors might be worth considering.

At the moment, window limited "`intercept`" charts have not been extensively tested and are at the moment not supported. As speed is not an issue here this doesn't bother too much. Therefore, a value of `M=-1` is always used in the intercept charts.

Value

`survRes` `algo.glrnb` returns a list of class `survRes` (surveillance result), which includes the alarm value for recognizing an outbreak (1 for alarm, 0 for no alarm), the threshold value for recognizing the alarm and the input object of class disProg. The `upperbound` slot of the object are filled with the current $GLR(n)$ value or with the number of cases that are necessary to produce an alarm at any timpoint $<= n$. Both lead to the same alarm timepoints, but `"cases"` has an obvious interpretation.

Author(s)

M. Hoehle

Source

Count data regression charts for the monitoring of surveillance time series (2008), M. Höhle and M. Paul, Computational Statistics and Data Analysis, 52(9), pp. 4357–4368.

Poisson regression charts for the monitoring of surveillance time series (2006), Höhle, M., SFB386 Discussion Paper 500.

See Also

algo.rkiLatestTimepoint

Examples

```
##Simulate data and apply the algorithm
S <- 1 ; t <- 1:120 ; m <- length(t)
beta <- c(1.5,0.6,0.6)
omega <- 2*pi/52
#log mu_{0,t}
alpha <- 0.2
base <- beta[1] + beta[2] * cos(omega*t) + beta[3] * sin(omega*t)
#Generate example data with changepoint and tau=tau
tau <- 100
kappa <- 0.4
mu0 <- exp(base)
mu1 <- exp(base  + kappa)

#Generate data
set.seed(42)
x <- rnbinom(length(t),mu=mu0*(exp(kappa)^(t>=tau)),size=1/alpha)
s.ts <- create.disProg(week=1:length(t),observed=x,state=(t>=tau))

#Plot the data
plot(s.ts,legend=NULL,xaxis.years=FALSE)

#Run GLR based detection
cntrl = list(range=t,c.ARL=5, Mtilde=1, mu0=mu0, alpha=alpha,
             change="intercept",ret="value",dir="inc")
glr.ts <- algo.glrnb(s.ts,control=c(cntrl))
plot(glr.ts,xaxis.years=FALSE)

#CUSUM LR detection with backcalculated number of cases
cntrl2 = list(range=t,c.ARL=5, Mtilde=1, mu0=mu0, alpha=alpha,
              change="intercept",ret="cases",dir="inc",theta=1.2)
glr.ts2 <- algo.glrnb(s.ts,control=c(cntrl2))
plot(glr.ts2,xaxis.years=FALSE)
```

---

algo.glrpois *Poisson regression charts*

Description

Poisson regression charts for the monitoring of surveillance time series.

Usage

```
algo.glrpois(disProgObj,control = list(range=range,c.ARL=5,
         mu0=NULL, Mtilde=1, M=-1, change="intercept",theta=NULL,
         dir=c("inc","dec"),ret=c("cases","value")))
```

Arguments

`disProgObj`	object of class `disProg` to do surveillance for
`control`	A list controlling the behaviour of the algorithm

`range` vector of indices in the observed vector to monitor (should be consecutive)

`mu0` A vector of in-control values of the Poisson distribution with the same length as `range`. If `NULL` the observed values in `1:(min(range)-1)` are used to estimate beta through a generalized linear model. To fine-tune the model one can instead specify `mu0` as a list with two components:

`S` number of harmonics to include

`trend` include a term `t` in the GLM model

`c.ARL` threshold in the GLR test, i.e. c_γ

`Mtilde` number of observations needed before we have a full rank the typical setup for the `"intercept"` and `"epi"` charts is `Mtilde=1`

`M` number of time instances back in time in the window-limited approach, i.e. the last value considered is $\max 1, n - M$. To always look back until the first observation use `M=-1`.

`change` a string specifying the type of the alternative. Currently the two choices are `intercept` and `epi`. See the SFB Discussion Paper 500 for details.

`theta` if `NULL` then the GLR scheme is used. If not `NULL` the prespecified value for κ or λ is used in a recursive LR scheme, which is faster.

`dir` a string specifying the direction of testing in GLR scheme. With `"inc"` only increases in x are considered in the GLR-statistic, with `"dec"` decreases are regarded.

`ret` a string specifying the type of `upperbound`-statistic that is returned. With `"cases"` the number of cases that would have been necassary to produce an alarm or with `"value"` the glr-statistic is computed (see below).

Details

This function implements the seasonal Poisson charts based on generalized likelihood ratio (GLR) as described in the SFB Discussion Paper 500. A moving-window generalized likelihood ratio detector is used, i.e. the detector has the form

$$N = \inf\left\{ n : \max_{1\leq k\leq n} \left[\sum_{t=k}^{n} \log\left\{ \frac{f_{\theta_1}(x_t|z_t)}{f_{\theta_0}(x_t|z_t)} \right\} \right] \geq c_\gamma \right\}$$

where instead of $1 \leq k \leq n$ the GLR statistic is computed for all $k \in \{n - M, \ldots, n - \tilde{M} + 1\}$. To achieve the typical behaviour from $1 \leq k \leq n$ use `Mtilde=1` and `M=-1`.

So N is the time point where the GLR statistic is above the threshold the first time: An alarm is given and the surveillance is resetted starting from time $N + 1$. Note that the same `c.ARL` as before is used, but if `mu0` is different at $N+1, N+2, \ldots$ compared to time $1, 2, \ldots$ the run length properties differ. Because `c.ARL` to obtain a specific ARL can only be obtained my Monte Carlo simulation there is no good way to update `c.ARL` automatically at the moment. Also, FIR GLR-detectors might be worth considering.

At the moment, window limited "`intercept`" charts have not been extensively tested and are at the moment not supported. As speed is not an issue here this doesn't bother too much. Therefore, a value of `M=-1` is always used in the intercept charts.

Value

`survRes` `algo.glrpois` returns a list of class `survRes` (surveillance result), which includes the alarm value for recognizing an outbreak (1 for alarm, 0 for no alarm), the threshold value for recognizing the alarm and the input object of class disProg. The `upperbound` slot of the object are filled with the current $GLR(n)$ value or with the number of cases that are necassary to produce an alarm at any timpoint $<= n$. Both lead to the same alarm timepoints, but `"cases"` has an obvious interpretation.

Author(s)

M. Hoehle with contributions by V. Wimmer

Source

Poisson regression charts for the monitoring of surveillance time series (2006), Höhle, M., SFB386 Discussion Paper 500.

See Also

`algo.rkiLatestTimepoint`

Examples

```
##Simulate data and apply the algorithm
S <- 1 ; t <- 1:120 ; m <- length(t)
beta <- c(1.5,0.6,0.6)
omega <- 2*pi/52
#log mu_{0,t}
base <- beta[1] + beta[2] * cos(omega*t) + beta[3] * sin(omega*t)
#Generate example data with changepoint and tau=tau
tau <- 100
kappa <- 0.4
mu0 <- exp(base)
mu1 <- exp(base  + kappa)

#Generate data
set.seed(42)
x <- rpois(length(t),mu0*(exp(kappa)^(t>=tau)))
s.ts <- create.disProg(week=1:length(t),observed=x,state=(t>=tau))
```

```
#Plot the data
plot(s.ts,legend=NULL,xaxis.years=FALSE)

#Run
cntrl = list(range=t,c.ARL=5, Mtilde=1, mu0=mu0,
             change="intercept",ret="value",dir="inc")
glr.ts <- algo.glrpois(s.ts,control=c(cntrl))
lr.ts  <- algo.glrpois(s.ts,control=c(cntrl,theta=0.4))

plot(glr.ts,xaxis.years=FALSE)
plot(lr.ts,xaxis.years=FALSE)
```

`algo.hhh` *Model fit based on the Held, Hoehle, Hofman paper*

Description

Fits a Poisson/negative binomial model with mean μ_{it} (as described in Held/Höhle/Hofmann, 2005) to a multivariate time series of counts.

Usage

```
algo.hhh(disProgObj, control=list(lambda=TRUE, neighbours=FALSE,
   linear=FALSE, nseason = 0,
   negbin=c("none", "single", "multiple"),
   proportion=c("none", "single", "multiple"),lag.range=NULL),
   thetastart=NULL, verbose=TRUE)
```

Arguments

`disProgObj`	object of class `disProg`
`control`	control object:

`lambda` If `TRUE` an autoregressive parameter λ is included, if `lambda` is a vector of logicals, unit-specific parameters λ_i are included. By default, observations y_{t-lag} at the previous time points, i.e. $lag = 1$, are used for the autoregression. Other lags can be used by specifying `lambda` as a vector of integers, see Examples and Details.

`neighbours` If `TRUE` an autoregressive parameter for adjacent units ϕ is included, if `neighbours` is a vector of logicals, unit-specific parameters ϕ_i are included. By default, observations y_{t-lag} at the previous time points, i.e. $lag = 1$, are used for the autoregression. Other lags can be used by specifying `neighbours` as a vector of integers.

`linear` a `logical` (or a vector of logicals) indicating wether a linear trend β (or a linear trend β_i for each unit) is included

`nseason` Integer number of Fourier frequencies; if `nseason` is a vector of integers, each unit i gets its own seasonal parameters

`negbin` if `"single"` negative binomial rather than poisson is used, if `"multiple"` unit-specific overdispersion parameters are used.

`proportion` see Details

`lag.range` determines which observations are used to fit the model

`thetastart` vector with starting values for all parameters specified in the control object (for `optim`).

`verbose` if `true` information about convergence is printed

Details

This functions fits a model as specified in equations (1.2) and (1.1) in Held et al. (2005) to univariate time series, and as specified in equations (3.3) and (3.2) (with extensions given in equations (2) and (4) in Paul et al., 2008) to multivariate time series.

For univariate time series, the mean structure of a Poisson or a negative binomial model is

$$\mu_t = \lambda y_{t-lag} + \nu_t$$

where

$$\log(\nu_t) = \alpha + \beta t + \sum_{j=1}^{S}(\gamma_{2j-1}\sin(\omega_j t) + \gamma_{2j}\cos(\omega_j t))$$

and $\omega_j = 2\pi j/period$ are Fourier frequencies with known period, e.g. `period=52` for weekly data.

Per default, the number of cases at time point $t-1$, i.e. $lag = 1$, enter as autoregressive covariates into the model. Other lags can also be considered.

For multivariate time series the mean structure is

$$\mu_{it} = \lambda_i y_{i,t-lag} + \phi_i \sum_{j\sim i} w_{ji} y_{j,t-lag} + n_{it}\nu_{it}$$

where

$$\log(\nu_{it}) = \alpha_i + \beta_i t + \sum_{j=1}^{S_i}(\gamma_{i,2j-1}\sin(\omega_j t) + \gamma_{i,2j}\cos(\omega_j t))$$

and n_{it} are standardized population counts. The weights w_{ji} are specified in the columns of the neighbourhood matrix `disProgObj$neighbourhood`.

Alternatively, the mean can be specified as

$$\mu_{it} = \lambda_i \pi_i y_{i,t-1} + \sum_{j\sim i} \lambda_j (1-\pi_j)/|k \sim j| y_{j,t-1} + n_{it}\nu_{it}$$

if `proportion="single"` ("multiple") in the `control` argument. Note that this model specification is still experimental.

Value

Returns an object of class ah with elements

coefficients	estimated parameters
se	estimated standard errors
cov	covariance matrix
loglikelihood	loglikelihood
convergence	logical indicating whether optim converged or not
fitted.values	fitted mean values $\mu_{i,t}$
control	specified control object
disProgObj	specified disProg-object
lag	which lag was used for the autoregressive parameters $lambda$ and phi
nObs	number of observations used for fitting the model

Note

For the time being this function is not a surveillance algorithm, but only a modelling approach as described in the papers by Held et. al (2005) and Paul et. al (2008).

Author(s)

M. Paul, L. Held, M. Höhle

References

Held, L., Höhle, M., Hofmann, M. (2005) A statistical framework for the analysis of multivariate infectious disease surveillance counts, Statistical Modelling, **5**, 187–199.

Paul, M., Held, L. and Toschke, A. M. (2008) Multivariate modelling of infectious disease surveillance data, Statistics in Medicine, **27**, 6250–6267.

See Also

algo.hhh.grid, hhh4

Examples

```
# univariate time series: salmonella agona cases
data(salmonella.agona)

model1 <- list(lambda=TRUE, linear=TRUE,
               nseason=1, negbin="single")

algo.hhh(salmonella.agona, control=model1)

# multivariate time series:
# measles cases in Lower Saxony, Germany
```

```
data(measles.weser)

# same model as above
algo.hhh(measles.weser, control=model1)

# include autoregressive parameter phi for adjacent "Kreise"
# specifiy start values for theta
model2 <- list(lambda = TRUE, neighbours = TRUE,
               linear = FALSE, nseason = 1,
               negbin = "single")
algo.hhh(measles.weser, control = model2, thetastart = rep(0, 20) )

## weekly counts of influenza and meningococcal infections
## in Germany, 2001-2006
data(influMen)

# specify model with two autoregressive parameters lambda_i, overdispersion
# parameters psi_i, an autoregressive parameter phi for meningococcal infections
# (i.e. nu_flu,t = lambda_flu * y_flu,t-1
#  and  nu_men,t = lambda_men * y_men,t-1 + phi_men*y_flu,t-1 )
# and S=(3,1) Fourier frequencies
model <- list(lambda=c(TRUE,TRUE), neighbours=c(FALSE,TRUE),
              linear=FALSE,nseason=c(3,1),negbin="multiple")

# run algo.hhh
algo.hhh(influMen, control=model)

# now meningococcal infections in the same week should enter as covariates
# (i.e. nu_flu,t = lambda_flu * y_flu,t-1
#  and  nu_men,t = lambda_men * y_men,t-1 + phi_men*y_flu,t )
model2 <- list(lambda=c(1,1), neighbours=c(NA,0),
              linear=FALSE,nseason=c(3,1),negbin="multiple")

algo.hhh(influMen, control=model2)
```

`algo.hhh.grid` *Function to try multiple starting values*

Description

Tries multiple starting values in `algo.hhh`. Starting values are provided in a matrix with `gridSize` rows, the grid search is conducted until either all starting values are used or a time limit `maxTime` is exceeded. The result with the highest likelihood is returned.

Usage

```
algo.hhh.grid(disProgObj, control=list(lambda=TRUE, neighbours=FALSE,
   linear=FALSE, nseason=0,
```

```
    negbin=c("none", "single", "multiple"),
    proportion=c("none", "single", "multiple"),lag.range=NULL),
    thetastartMatrix, maxTime=1800, verbose=FALSE)
```

Arguments

disProgObj	object of class disProg
control	control object:
	lambda If TRUE an autoregressive parameter λ is included, if lambda is a vector of logicals, unit-specific parameters λ_i are included. By default, observations y_{t-lag} at the previous time points, i.e. $lag = 1$, are used for the autoregression. Other lags can be used by specifying lambda as a vector of integers, see Examples and algo.hhh for details.
	neighbours If TRUE an autoregressive parameter for adjacent units ϕ is included, if neighbours is a vector of logicals, unit-specific parameters ϕ_i are included. By default, observations y_{t-lag} at the previous time points, i.e. $lag = 1$, are used for the autoregression. Other lags can be used by specifying neighbours as a vector of integers.
	linear a logical (or a vector of logicals) indicating wether a linear trend β (or a linear trend β_i for each unit) is included
	nseason integer number of Fourier frequencies; if nseason is a vector of integers, each unit i gets its own seasonal parameters
	negbin if "single" negative binomial rather than poisson is used, if "multiple" unit-specific overdispersion parameters are used.
	proportion see details in algo.hhh
	lag.range determines which observations are used to fit the model
thetastartMatrix	matrix with initial values for all parameters specified in the control object as rows.
verbose	if true progress information is printed
maxTime	maximum of time (in seconds) to elapse until algorithm stopps.

Value

Returns an object of class ahg with elements

best	result of a call to algo.hhh with highest likelihood
allLoglik	values of loglikelihood for all starting values used
gridSize	number of different starting values in thetastartMatrix
gridUsed	number of used starting values
time	elapsed time
convergence	if false algo.hhh did not converge for all (used) starting values

Author(s)

M. Paul, L. Held

References

Held, L., Höhle, M., Hofmann, M. (2005) A statistical framework for the analysis of multivariate infectious disease surveillance counts, Statistical Modelling, **5**, 187–199.

Paul, M., Held, L. and Toschke, A. M. (2008) Multivariate modelling of infectious disease surveillance data, Statistics in Medicine, **27**, 6250–6267.

See Also

algo.hhh, create.grid

Examples

```
## Not run:
## monthly counts of menigococcal infections in France
data(meningo.age)

# specify model for algo.hhh.grid
model1 <- list(lambda=TRUE)

# create grid of inital values
grid1 <- create.grid(meningo.age, model1,
                     params = list(epidemic=c(0.1,0.9,5)))

# try multiple starting values, print progress information
algo.hhh.grid(meningo.age, control=model1, thetastartMatrix=grid1,
               verbose=TRUE)

# specify model
model2 <- list(lambda=TRUE, neighbours=TRUE, negbin="single",
               nseason=1)
grid2 <- create.grid(meningo.age, model2,
                     params = list(epidemic=c(0.1,0.9,3),
                                   endemic=c(-0.5,0.5,3),
                                   negbin = c(0.3, 12, 10)))

# run algo.hhh.grid, search time is limited to 30 sec
algo.hhh.grid(meningo.age, control=model2, thetastartMatrix=grid2,
              maxTime=30)

## weekly counts of influenza and meningococcal infections in Germany, 2001-2006
data(influMen)

# specify model with two autoregressive parameters lambda_i, overdispersion
# parameters psi_i, an autoregressive parameter phi for meningococcal infections
# (i.e. nu_flu,t = lambda_flu * y_flu,t-1
#  and  nu_men,t = lambda_men * y_men,t-1 + phi_men*y_flu,t-1 )
# and S=(3,1) Fourier frequencies
model <- list(lambda=c(TRUE,TRUE), neighbours=c(FALSE,TRUE),
              linear=FALSE, nseason=c(3,1),negbin="multiple")

# create grid of initial values
```

```
grid <- create.grid(influMen,model, list(epidemic=c(.1,.9,3),
             endemic=c(-.5,.5,3), negbin=c(.3,15,10)))
# run algo.hhh.grid, search time is limited to 30 sec
algo.hhh.grid(influMen, control=model, thetastartMatrix=grid, maxTime=30)

# now meningococcal infections in the same week should enter as covariates
# (i.e. nu_flu,t = lambda_flu * y_flu,t-1
#  and  nu_men,t = lambda_men * y_men,t-1 + phi_men*y_flu,t )
model2 <- list(lambda=c(1,1), neighbours=c(NA,0),
             linear=FALSE,nseason=c(3,1),negbin="multiple")

algo.hhh.grid(influMen, control=model2, thetastartMatrix=grid, maxTime=30)

## End(Not run)
```

`algo.hmm` *Hidden Markov Model (HMM) method*

Description

This function implements on-line HMM detection of outbreaks based on the retrospective procedure described in Le Strat and Carret (1999). Using the function `msm` (from package **msm**) a specified HMM is estimated, the decoding problem, i.e. the most probable state configuration, is found by the Viterbi algorithm and the most probable state of the last observation is recorded. On-line detection is performed by sequentally repeating this procedure.

Warning: This function can be very slow - a more efficient implementation would be nice!

Usage

```
algo.hmm(disProgObj, control = list(range=range, Mtilde=-1,
         noStates=2, trend=TRUE, noHarmonics=1,
         covEffectEqual=FALSE, saveHMMs = FALSE, extraMSMargs=list()))
```

Arguments

`disProgObj`	object of class disProg (including the observed and the state chain)
`control`	control object:
	`range` determines the desired time points which should be evaluated. Note that opposite to other surveillance methods an initial parameter estimation occurs in the HMM. Note that range should be high enough to allow for enough reference values for estimating the HMM
	`Mtilde` number of observations back in time to use for fitting the HMM (including the current observation). Reasonable values are a multiple of `disProgObj$freq`, the default is `Mtilde=-1`, which means to use all possible values - for long series this might take very long time!
	`noStates` number of hidden states in the HMM – the typical choice is 2. The initial rates are set such that the `noStates`'th state is the one having the highest rate. In other words: this state is considered the outbreak state.

`trend` Boolean stating whether a linear time trend exists, i.e. if `TRUE` (default) then $\beta_j \neq 0$

`noHarmonics` number of harmonic waves to include in the linear predictor. Default is 1.

`covEffectEqual` see details

`saveHMMs` Boolean, if `TRUE` then the result of the fitted HMMs is saved. With this option the function can also be used to analyse data retrospectively. Default option is `FALSE`

`extraMSMArgs` A named list with additional arguments to send to the `msm` HMM fitting function. Note that the `msm` arguments `formula`, `data`, `qmatrix`, `hmodel`, `hcovariates` and `hconstraint` are automatically filled by `algo.hmm`, thus these should NOT be modified.

Details

For each time point t the reference values values are extracted. If the number of requested values is larger than the number of possible values the latter is used. Now the following happens on these reference values:

A `noState`-State Hidden Markov Model (HMM) is used based on the Poisson distribution with linear predictor on the log-link scale. I.e.

$$Y_t | X_t = j \sim Po(\mu_t^j),$$

where

$$\log(\mu_t^j) = \alpha_j + \beta_j \cdot t + \sum_{i=1}^{nH} \gamma_j^i \cos(2i\pi/freq \cdot (t-1)) + \delta_j^i \sin(2i\pi/freq \cdot (t-1))$$

and nH =`noHarmonics` and $freq = 12, 52$ depending on the sampling frequency of the surveillance data. In the above $t-1$ is used, because the first week is always saved as `t=1`, i.e. we want to ensure that the first observation corresponds to cos(0) and sin(0).

If `covEffectEqual` then all covariate effects parameters are equal for the states, i.e. $\beta_j = \beta, \gamma_j^i = \gamma^i, \delta_j^i = \delta^i$ for all $j = 1, ..., noState$.

In case more complicated HMM models are to be fitted it is possible to modify the `msm` code used in this function. Using e.g. `AIC` one can select between different models (see the **msm** package for further details).

Using the Viterbi algorithms the most probable state configuration is obtained for the reference values and if the most probable configuration for the last reference value (i.e. time t) equals `control$noOfStates` then an alarm is given.

Note: The HMM is re-fitted from scratch every time, sequential updating schemes of the HMM would increase speed considerably! A major advantage of the approach is that outbreaks in the reference values are handled automatically.

Value

`survRes` `algo.hmm` gives a list of class `survRes` which includes the vector of alarm values for every timepoint in `range`. No `upperbound` can be specified and is put equal to zero.

The resulting object contains a slot `control$hmm`, which contains the `msm` object with the fitted HMM.

Author(s)

M. Höhle

References

Y. Le Strat and F. Carrat, Monitoring Epidemiologic Surveillance Data using Hidden Markov Models (1999), Statistics in Medicine, 18, 3463–3478

I.L. MacDonald and W. Zucchini, Hidden Markov and Other Models for Discrete-valued Time Series, (1997), Chapman & Hall, Monographs on Statistics and applied Probability 70

See Also

`msm`

Examples

```
## Not run:
library("msm")
set.seed(123)
#Simulate outbreak data from HMM
counts <- sim.pointSource(p = 0.98, r = 0.8, length = 3*52,
                              A = 1, alpha = 1, beta = 0, phi = 0,
                              frequency = 1, state = NULL, K = 1.5)

#Do surveillance using a two state HMM without trend component and
#the effect of the harmonics being the same in both states. A sliding
#window of two years is used to fit the HMM
surv <- algo.hmm(counts, control=list(range=(2*52):length(counts$observed),
                                  Mtilde=2*52,noStates=2,trend=FALSE,
                                  covEffectsEqual=TRUE,extraMSMargs=list()))
plot(surv,legend=list(x="topright"))

#Retrospective use of the function, i.e. monitor only the last time point
#but use option saveHMMs to store the output of the HMM fitting
surv <- algo.hmm(counts,control=list(range=length(counts$observed),Mtilde=-1,noStates=2,
                          trend=FALSE,covEffectsEqual=TRUE, saveHMMs=TRUE))

#Compute most probable state using the viterbi algorithm - 1 is "normal", 2 is "outbreak".
viterbi.msm(surv$control$hmm[[1]])$fitted

#How often correct?
tab <- cbind( truth=counts$state + 1 , hmm=viterbi.msm(surv$control$hmm[[1]])$fitted)
table(tab[,1],tab[,2])

## End(Not run)
```

`algo.outbreakP` *Semiparametric surveillance of outbreaks*

Description

Frisen and Andersson (2009) method for semiparametric surveillance of outbreaks

Usage

```
algo.outbreakP(disProgObj, control = list(range = range, k=100,
               ret=c("cases","value"),maxUpperboundCases=1e5))
```

Arguments

`disProgObj`	object of class disProg (including the observed and the state chain).
`control`	A list controlling the behaviour of the algorithm
	`range` determines the desired time-points which should be monitored. Note that it is automatically assumed that ALL other values in `disProgObj` can be used for the estimation, i.e. for a specific value `i` in `range` all values from 1 to `i` are used for estimation.
	`k` The threshold value. Once the outbreak statistic is above this threshold `k` an alarm is sounded.
	`ret` a string specifying the type of `upperbound`-statistic that is returned. With `"cases"` the number of cases that would have been necessary to produce an alarm (NNBA) or with `"value"` the outbreakP-statistic is computed (see below).
	`maxUpperboundCases` Upperbound when numerically searching for NNBA. Default is 1e5.

Details

A generalized likelihood ratio test based on the Poisson distribution is implemented where the means of the in-control and out-of-control states are computed by isotonic regression.

$$OutbreakP(s) = \prod_{t=1}^{s} \left(\frac{\hat{\mu}^{C1}(t)}{\hat{\mu}^{D}(t)} \right)^{x(t)}$$

,

where $\hat{\mu}^{C1}(t)$ is the estimated mean obtained by uni-modal regression under the assumption of one change-point and $\hat{\mu}^{D}(t)$ is the estimated result when there is no change-point (i.e. this is just the mean of all observations). Note that the contrasted hypothesis assume all means are equal until the change-point, i.e. this detection method is especially suited for detecting a shift from a relative constant mean. Hence, this is less suited for detection in diseases with strong seasonal endemic component. Onset of influenza detection is an example where this method works particular well.

In case `control$ret == "cases"` then a brute force numerical search for the number needed before alarm (NNBA) is performed. That is, given the past observations, whats the minimum number

which would have caused an alarm? Note: Computing this might take a while because the search is done by sequentially increasing/decreasing the last observation by one for each time point in `control$range` and then calling the workhorse function of the algorithm again. The argument `control$maxUpperboundCases` controls the upper limit of this search (default is 1e5). Currently, even though the statistic has passed the threshold, the NNBA is still computed. After a few time instances what typically happens is that no matter the observed value we would have an alarm at this time point. In this case the value of NNBA is set to `NA`. Furthermore, the first time point is always `NA`, unless `k<1`.

Value

`survRes` `algo.outbreakP` gives a list of class `survRes` which includes the vector of alarm values for every time-point in `range`, the vector of threshold values for every time-point in `range`.

Author(s)

M. Höhle – based on Java code by Frisen and Schiöler

Source

The code is an extended R port of the Java code by Marianne Frisén and Linus Schiöler from the CASE project available under the GNU GPL License v3. See https://smisvn.smi.se/case/ for further details on the CASE project. A manual on how to use an Excel implementation of the method is available at http://www.hgu.gu.se/item.aspx?id=16857.

The R code contains e.g. the search for NNBA (see details).

References

Frisén, Andersson and Schiöler (2009), Robust outbreak surveillance of epidemics in Sweden, Statistics in Medicine, 28(3):476-493.

Frisén and Andersson (2009) Semiparametric Surveillance of Monotonic Changes, Sequential Analysis 28(4):434-454.

Examples

```
#Use data from outbreakP manual (http://www.hgu.gu.se/item.aspx?id=16857)
y <- matrix(c(1,0,3,1,2,3,5,4,7,3,5,8,16,23,33,34,48),ncol=1)

#Generate sts object with these observations
mysts <- new("sts", observed=y, epoch=1:length(y), alarm=y*0,
                    start=c(2000,1), freq=52)

#Run the algorithm and present results
#Only the value of outbreakP statistic
upperbound(outbreakP(mysts, control=list(range=1:length(y),k=100,
           ret="value")))

#Graphical illustration with number-needed-before-alarm (NNBA) upperbound.
res <- outbreakP(mysts, control=list(range=1:length(y),k=100,
```

```
            ret="cases"))
plot(res,dx.upperbound=0,lwd=c(1,1,3),legend.opts=list(legend=c("Infected",
      "NNBA","Outbreak","Alarm"),horiz=TRUE))
```

`algo.quality` *Computation of Quality Values for a Surveillance System Result*

Description

Computation of the quality values for a surveillance System output.

Usage

```
algo.quality(survResObj, penalty = 20)
```

Arguments

`survResObj`	object of class survRes, which includes the state chain and the computed alarm chain
`penalty`	the maximal penalty for the lag

Details

The lag is defined as follows: In the state chain just the beginnings of an outbreak chain (outbreaks directly following each other) are considered. In the alarm chain, the range from the beginning of an outbreak until $min(nextoutbreakbeginning, \texttt{penalty})$ timepoints is considered. The `penalty` timepoints were chosen, to provide an upper bound on the penalty for not discovering an outbreak. Now the difference between the first alarm by the system and the defined beginning is denoted "the lag" Additionally outbreaks found by the system are not punished. At the end, the mean of the lags for every outbreak chain is returned as summary lag.

Value

`list of quality values`

- TP: Number of correct found outbreaks.
- FP: Number of false found outbreaks.
- TN: Number of correct found non outbreaks.
- FN: Number of false found non outbreaks.
- sens: True positive rate, meaning TP/(FN + TP).
- spec: True negative rate, meaning TN/(TN + FP).
- dist: Euclidean distance between (1-spec, sens) to (0,1).
- lag: Lag of the outbreak recognizing by the system.

See Also

`algo.compare`

Examples

```
# Create a test object
disProgObj <- sim.pointSource(p = 0.99, r = 0.5, length = 200, A = 1,
                                  alpha = 1, beta = 0, phi = 0,
                                  frequency = 1, state = NULL, K = 1.7)

# Let this object be tested from rki1
survResObj <- algo.rki1(disProgObj, control = list(range = 50:200))

# Compute the quality values
algo.quality(survResObj)
```

`algo.rki` *The system used at the RKI*

Description

Evaluation of timepoints with the detection algorithms used by the RKI

Usage

```
algo.rkiLatestTimepoint(disProgObj, timePoint = NULL,
     control = list(b = 2, w = 4, actY = FALSE))
algo.rki(disProgObj, control = list(range = range,
     b = 2, w = 4, actY = FALSE))
algo.rki1(disProgObj, control = list(range = range))
algo.rki2(disProgObj, control = list(range = range))
algo.rki3(disProgObj, control = list(range = range))
```

Arguments

`disProgObj`	object of class disProg (including the observed and the state chain).
`timePoint`	time point which shoud be evaluated in `algo.rkiLatestTimepoint`. The default is to use the latest timepoint.
`control`	control object: `range` determines the desired timepoints which should be evaluated, `b` describes the number of years to go back for the reference values, `w` is the half window width for the reference values around the appropriate timepoint and `actY` is a boolean to decide if the year of `timePoint` also spend `w` reference values of the past. As default `b`, `w`, `actY` are set for the RKI 3 system.

Details

Using the reference values for calculating an upper limit (threshold), alarm is given if the actual value is bigger than a computed threshold. `algo.rki` calls `algo.rkiLatestTimepoint` for the values specified in `range` and for the system specified in `control`. `algo.rki1` calls `algo.rkiLatestTimepoint` for the values specified in `range` for the RKI 1 system. `algo.rki2` calls `algo.rkiLatestTimepoint` for the values specified in `range` for the RKI 2 system. `algo.rki3` calls `algo.rkiLatestTimepoint` for the values specified in `range` for the RKI 3 system.

- "RKI 1" reference values from 6 weeks ago
- "RKI 2" reference values from 6 weeks ago and 13 weeks of the year ago (symmetrical around the comparable week).
- "RKI 3" 18 reference values. 9 from the year ago and 9 from two years ago (also symmetrical around the comparable week).

Value

survRes — algo.rkiLatestTimepoint returns a list of class survRes (surveillance result), which includes the alarm value (alarm = 1, no alarm = 0) for recognizing an outbreak, the threshold value for recognizing the alarm and the input object of class disProg.

algo.rki gives a list of class survRes which includes the vector of alarm values for every timepoint in range, the vector of threshold values for every timepoint in range for the system specified by b, w and actY, the range and the input object of class disProg. algo.rki1 returns the same for the RKI 1 system, algo.rki2 for the RKI 2 system and algo.rki3 for the RKI 3 system.

Author(s)

M. Höhle, A. Riebler, Christian Lang

See Also

algo.bayesLatestTimepoint and algo.bayes for the Bayes system.

Examples

```
# Create a test object
disProgObj <- sim.pointSource(p = 0.99, r = 0.5, length = 208, A = 1,
                                  alpha = 1, beta = 0, phi = 0,
                                  frequency = 1, state = NULL, K = 1.7)

# Test week 200 to 208 for outbreaks with a selfdefined rki
algo.rki(disProgObj, control = list(range = 200:208, b = 1,
                                        w = 5, actY = TRUE))
# The same for rki 1 to rki 3
algo.rki1(disProgObj, control = list(range = 200:208))
algo.rki2(disProgObj, control = list(range = 200:208))
algo.rki3(disProgObj, control = list(range = 200:208))

# Test for rki 1 the latest timepoint
algo.rkiLatestTimepoint(disProgObj)
```

`algo.rogerson` *Modified CUSUM method as proposed by Rogerson and Yamada (2004)*

Description

Modified Poisson CUSUM method that allows for a time-varying in-control parameter $\theta_{0,t}$ as proposed by Rogerson and Yamada (2004). The same approach can be applied to binomial data if `distribution="binomial"` is specified.

Usage

```
algo.rogerson(disProgObj, control = list(range = range,
   theta0t = NULL, ARL0 = NULL, s = NULL, hValues = NULL,
   distribution = c("poisson","binomial"), nt = NULL, FIR=FALSE,
   limit = NULL, digits = 1))
```

Arguments

`disProgObj`	object of class `disProg` that includes a matrix with the observed number of counts
`control`	list with elements

range vector of indices in the observed matrix of `disProgObj` to monitor

theta0t matrix with in-control parameter, must be specified

ARL0 desired average run length γ

s change to detect, see `findH` for further details

hValues matrix with decision intervals `h` for a sequence of values $\theta_{0,t}$ (in the range of `theta0t`)

distribution `"poisson"` or `"binomial"`

nt optional matrix with varying sample sizes for the binomial CUSUM

FIR a FIR CUSUM with head start $\frac{\mathsf{h}}{2}$ is applied to the data if `TRUE`, otherwise no head start is used; see details

limit numeric that determines the procedure after an alarm is given, see details

digits the reference value and decision interval are rounded to `digits` decimal places. Defaults to 1 and should correspond to the number of digits used to compute `hValues`

Details

The CUSUM for a sequence of Poisson or binomial variates x_t is computed as

$$S_t = \max\{0, S_{t-1} + c_t(x_t - k_t)\},\ t = 1, 2, \ldots,$$

where $S_0 = 0$ and $c_t = \frac{h}{h_t}$; k_t and h_t are time-varying reference values and decision intervals. An alarm is given at time t if $S_t \geq h$.

If FIR=TRUE, the CUSUM starts with a head start value $S_0 = \frac{h}{2}$ at time $t = 0$. After an alarm is given, the FIR CUSUM starts again at this head start value.

The procedure after the CUSUM gives an alarm can be determined by limit. Suppose that the CUSUM signals at time t, i.e. $S_t \geq h$.

For numeric values of limit, the CUSUM is bounded above after an alarm is given, i.e. S_t is set to $\min\{\texttt{limit} \cdot h, S_t\}$.

Using limit=0 corresponds to resetting S_t to zero after an alarm as proposed in the original formulation of the CUSUM. If FIR=TRUE, S_t is reset to $\frac{h}{2}$ (i.e. limit=$\frac{h}{2}$). If limit=NULL, no resetting occurs after an alarm is given.

Value

Returns an object of class survRes with elements

alarm	indicates whether the CUSUM signaled at time t or not (1 = alarm, 0 = no alarm)
upperbound	CUSUM values S_t
disProgObj	disProg object
control	list with the alarm threshold h and the specified control object

Note

algo.rogerson is a univariate CUSUM method. If the data are available in several regions (i.e. observed is a matrix), multiple univariate CUSUMs are applied to each region.

References

Rogerson, P. A. and Yamada, I. Approaches to Syndromic Surveillance When Data Consist of Small Regional Counts. Morbidity and Mortality Weekly Report, 2004, 53/Supplement, 79-85

See Also

hValues

Examples

```
# simulate data
set.seed(123)
data <- simHHH(control = list(coefs = list(alpha =-0.5, gamma = 0.4,
               delta = 0.6)),length=300)

# extract mean used to generate the data
lambda <- data$endemic

# determine a matrix with h values
hVals <- hValues(theta0 = 10:150/100, ARL0=500, s = 1, distr = "poisson")

# apply modified Poisson CUSUM
res <- algo.rogerson(data$data,
            control=c(hVals, list(theta0t=lambda,range=1:300)))
plot(res)
```

algo.summary *Summary Table Generation for Several Disease Chains*

Description

Summary table generation for several disease chains.

Usage

```
algo.summary(compMatrices)
```

Arguments

compMatrices	list of matrices constructed by algo.compare.

Details

As lag the mean of all single lags is returned. TP values, FN values, TN values and FP values are summed up. dist, sens and spec are new computed on the basis of the new TP value, FN value, TN value and FP value.

Value

matrix	summing up the singular input matrices

See Also

algo.compare, algo.quality

Examples

```
# Create a test object
disProgObj1 <- sim.pointSource(p = 0.99, r = 0.5, length = 400,
                        A = 1, alpha = 1, beta = 0, phi = 0,
                        frequency = 1, state = NULL, K = 1.7)
disProgObj2 <- sim.pointSource(p = 0.99, r = 0.5, length = 400,
                        A = 1, alpha = 1, beta = 0, phi = 0,
                        frequency = 1, state = NULL, K = 5)
disProgObj3 <- sim.pointSource(p = 0.99, r = 0.5, length = 400,
                        A = 1, alpha = 1, beta = 0, phi = 0,
                        frequency = 1, state = NULL, K = 17)

# Let this object be tested from any methods in range = 200:400
range <- 200:400
control <- list( list(funcName = "rki1", range = range),
                 list(funcName = "rki2", range = range),
                 list(funcName = "rki3", range = range)
               )
```

```
compMatrix1 <- algo.compare(algo.call(disProgObj1, control=control))
compMatrix2 <- algo.compare(algo.call(disProgObj2, control=control))
compMatrix3 <- algo.compare(algo.call(disProgObj3, control=control))

algo.summary( list(a=compMatrix1, b=compMatrix2, c=compMatrix3) )
```

`algo.twins` *Model fit based on a two-component epidemic model*

Description

Fits a negative binomial model (as described in Held et al. (2006) to an univariate time series of counts.

Usage

```
algo.twins(disProgObj, control=list(burnin=1000, filter=10,
   sampleSize=2500, noOfHarmonics=1, alpha_xi=10, beta_xi=10,
   psiRWSigma=0.25,alpha_psi=1, beta_psi=0.1, nu_trend=FALSE,
   logFile="twins.log"))
```

Arguments

`disProgObj` object of class `disProg`

`control` control object:

`burnin` Number of burn in samples.

`filter` Thinning parameter. If `filter = 10` every 10th sample is after the burn in is returned.

`sampleSize` Number of returned samples. Total number of samples = `burnin+filter*sampleSize`

`noOfHarmonics` Number of harmonics to use in the modelling, i.e. L in (2.2) of Held et al (2006).

`alpha_xi` Parameter α_ξ of the hyperprior of the epidemic parameter λ

`beta_xi` Parameter β_ξ of the hyperprior of the epidemic parameter λ

`psiRWSigma` Starting value for the tuning of the variance of the random walk proposal for the overdispersion parameter ψ.

`alpha_psi` Parameter α_ψ of the prior of the overdispersion parameter ψ

`beta_psi` Parameter β_ψ of the prior of the overdispersion parameter ψ

`nu_trend` Adjust for a linear trend in the endemic part? (default: `FALSE`)

`logFile` Base file name for the output files. The function writes three output files in your current working directory (i.e. getwd()). If `logfile = "twins.log"` the results are stored in the three files "twins.log", "twins.log2" and "twins.log.acc". "twins.log" containes the returned samples of the parameters ψ, γ_0, γ_1, γ_2, K, ξ_λ $\lambda_1, ..., \lambda n$, the predictive distribution of the number of cases at time $n + 1$ and the deviance. "twins.log2" containes the sample means of the variables X_t, Y_t, ω_t and the relative frequency of a changepoint at time t for

t=1,...,n and the relative frequency of a predicted changepoint at time n+1. "twins.log.acc" contains the acceptance rates of ψ, the changepoints and the endemic parameters γ_0, γ_1, γ_2 in the third column and the variance of the random walk proposal for the update of the parameter ψ in the second column.

Details

Note that for the time being this function is not a surveillance algorithm, but only a modelling approach as described in the Held et. al (2006) paper.

Note also that the function writes three logfiles in your current working directory (i.e. getwd()): twins.log, twins.log.acc and twins.log2 Thus you need to have write permissions in the current getwd() directory.

Value

Returns an object of class `atwins` with elements

`control`	specified control object
`disProgObj`	specified `disProg`-object
`logFile`	containes the returned samples of the parameters ψ, γ_0, γ_1, γ_2, K, ξ_λ $\lambda_1, ..., \lambda n$, the predictive distribution and the deviance.
`logFile2`	containes the sample means of the variables X_t, Y_t, ω_t and the relative frequency of a changepoint at time t for t=1,...,n and the relative frequency of a predicted changepoint at time n+1.

Author(s)

M. Hofmann and M. Höhle and D. Sabanés Bové

References

Held, L., Hofmann, M., Höhle, M. and Schmid V. (2006): A two-component model for counts of infectious diseases. *Biostatistics*, **7**, pp. 422–437.

Examples

```
# Load the data used in the Held et al. (2006) paper
data("hepatitisA")

# Fix seed - this is used for the MCMC samplers in twins
set.seed(123)

# Call algorithm and save result (use short chain without filtering for speed)
otwins <- algo.twins(hepatitisA,
                     control=list(burnin=500, filter=1, sampleSize=1000))

# This shows the entire output (use ask=TRUE for pause between plots)
plot(otwins, ask=FALSE)
```

```
# Direct access to MCMC output
hist(otwins$logFile$psi,xlab=expression(psi),main="")
if (require("coda")) {
    print(summary(mcmc(otwins$logFile[,c("psi","xipsi","K")])))
}
```

`animate` *Generic animation of spatio-temporal objects*

Description

Generic function for animation of R objects.

Usage

```
animate(object, ...)
```

Arguments

`object`	The object to animate.
`...`	Arguments to be passed to methods, such as graphical parameters or time interval options for the snapshots.

See Also

The methods `animate.epidata` and `animate.epidataCS` for the animation of epidemics.

`anscombe.residuals` *Compute Anscombe residuals*

Description

The residuals of `m` are transformed to form Anscombe residuals. which makes them approximately standard normal distributed.

Usage

```
anscombe.residuals(m, phi)
```

Arguments

`m`	`m` is a glm object of the fit
`phi`	`phi` is the current estimated over-dispersion

Value

Standardized Anscombe residuals of `m`

References

McCullagh & Nelder, Generalized Linear Models, 1989

`arlCusum` *Calculation of Average Run Length for discrete CUSUM schemes*

Description

Calculates the average run length (ARL) for an upward CUSUM scheme for discrete distributions (i.e. Poisson and binomial) using the Markov chain approach.

Usage

```
arlCusum(h=10, k=3, theta=2.4, distr=c("poisson","binomial"),
         W=NULL, digits=1, ...)
```

Arguments

`h`	decision interval
`k`	reference value
`theta`	distribution parameter for the cumulative distribution function (cdf) F, i.e. rate λ for Poisson variates or probability p for binomial variates
`distr`	`"poisson"` or `"binomial"`
`W`	Winsorizing value `W` for a robust CUSUM, to get a nonrobust CUSUM set `W` > `k+h`. If `NULL`, a nonrobust CUSUM is used.
`digits`	`k` and `h` are rounded to `digits` decimal places
`...`	further arguments for the distribution function, i.e. number of trials `n` for binomial cdf

Value

Returns a list with the ARL of the regular (zero-start) and the fast initial response (FIR) CUSUM scheme with reference value `k`, decision interval `h` for $X \sim F(\theta)$, where F is the Poisson or binomial cdf

`ARL`	one-sided ARL of the regular (zero-start) CUSUM scheme
`FIR.ARL`	one-sided ARL of the FIR CUSUM scheme with head start $\frac{h}{2}$

Source

Based on the FORTRAN code of

Hawkins, D. M. (1992). Evaluation of Average Run Lengths of Cumulative Sum Charts for an Arbitrary Data Distribution. Communications in Statistics - Simulation and Computation, 21(4), p. 1001-1020.

backprojNP *Non-parametric back-projection of incidence cases to exposure cases using a known incubation time as in Becker et al (1991).*

Description

The function is an implementation of the non-parametric back-projection of incidence cases to exposure cases described in Becker et al. (1991). The method back-projects exposure times from a univariate time series containing the number of symptom onsets per time unit. Here, the delay between exposure and symptom onset for an individual is seen as a realization of a random variable governed by a known probability mass function. The back-projection function calculates the expected number of exposures λ_t for each time unit under the assumption of a Poisson distribution, but without any parametric assumption on how the λ_t evolve in time.

Furthermore, the function contains a boostrap based procedure, as given in Yip et al (2011), which allows an indication of uncertainty in the estimated λ_t. The procedure is equivalent to the suggestion in Becker and Marschner (1993). However, the present implementation in `backprojNP` allows only a univariate time series, i.e. simulteneous age groups as in Becker and Marschner (1993) are not possible.

The method in Becker et al. (1991) was originally developed for the back-projection of AIDS incidence, but it is equally useful for analysing the epidemic curve in outbreak situations of a disease with long incubation time, e.g. in order to qualitatively investigate the effect of intervention measures.

Usage

```
backprojNP(sts, incu.pmf,
   control = list(k = 2,
                  eps = rep(0.005,2),
                  iter.max=rep(250,2),
                  Tmark = nrow(sts),
                  B = -1,
                  alpha = 0.05,
                  verbose = FALSE,
                  lambda0 = NULL,
                  eq3a.method = c("R","C"),
                  hookFun = function(stsbp) {}),
     ...)
```

Arguments

`sts`	an object of class "`sts`" (or one that can be coerced to that class): contains the observed number of symptom onsets as a time series.
`incu.pmf`	Probability mass function (PMF) of the incubation time. The PMF is specified as a vector or matrix with the value of the PMF evaluated at $0, ..., d_max$, i.e. note that the support includes zero. The value of d_max is automatically calculated as `length(incu.pmf)-1` or `nrow(incu.pmf)-1`. Note that if the sts object has

more than one column, then for the backprojection the incubation time is either recycled for all components or, if it is a matrix with the same number of columns as the sts object, the k'th column of `incu.pmf` is used for the backprojection of the k'th series.

`control` A list with named arguments controlling the functionality of the non-parametric back-projection.

`k` An integer representing the smoothing parameter to use in the smoothing step of the EMS algorithm. Needs to be an even number.

`eps` A vector of length two representing the convergence threshold ϵ of the EMS algorithm, see Details for further information. The first value is the threshold to use in the $k = 0$ loop, which forms the values for the parametric bootstrap. The second value is the threshold to use in the actual fit and bootstrap fitting using the specified `k`. If `k` is only of length one, then this number is replicated twice.

`Tmark` Numeric with $T' \leq T$. Upper time limit on which to base convergence, i.e. only the values $\lambda_1, \ldots, \lambda_{T'}$ are monitored for convergence. See details.

`iter.max` The maximum number of EM iterations to do before stopping.

`B` Number of parametric bootstrap samples to perform from an initial k=0 fit. For each sample a back projection is performed. See Becker and Marschner (1993) for details.

`alpha` (1-α)*100% confidence intervals are computed based on the percentile method.

`verbose` (boolean). If true show extra progress and debug information.

`lambda0` Start values for lambda. Vector needs to be of the length `nrow(sts)`.

`eq3a.method` A single character being either `"R"` or `"C"` depending on whether the three nested loops of equation 3a in Becker et al. (1991) are to be executed as safe R code (can be extremely slow, however the implementation is not optimized for speed) or a C code (can be more than 200 times faster!). However, the C implementation is experimental and can hang R if, e.g., the time series does not go far enough back.

`hookFun` Hook function called for each iteration of the EM algorithm. The function should take a single argument `stsbp` of class `"stsBP"` class. It will be have the lambda set to the current value of lambda. If no action desired just leave the function body empty (default). Additional arguments are possible.

`...` Additional arguments are sent to the hook function.

Details

Becker et al. (1991) specify a non-parametric back-projection algorithm based on the Expectation-Maximization-Smoothing (EMS) algorithm.

In the present implementation the algorithm iterates until

$$\frac{||\lambda^{(k+1)} - \lambda^{(k)}||}{||\lambda^{(k)}||} < \epsilon$$

This is a slight adaptation of the proposals in Becker et al. (1991). If T is the length of λ then one can avoid instability of the algorithm near the end by considering only the λ's with index $1, \ldots, T'$.

See the references for further information.

Value

backprojNP returns an object of "stsBP".

Note

The method is still experimental. A proper plot routine for stsBP objects is currently missing.

Author(s)

Michael Höhle with help by Daniel Sabanés Bové for the **Rcpp** interface

References

Becker NG, Watson LF and Carlin JB (1991), A method for non-parametric back-projection and its application to AIDS data, Statistics in Medicine, 10:1527-1542.

Becker NG and Marschner IC (1993), A method for estimating the age-specific relative risk of HIV infection from AIDS incidence data, Biometrika, 80(1):165-178.

Yip PSF, Lam KF, Xu Y, Chau PH, Xu J, Chang W, Peng Y, Liu Z, Xie X and Lau HY (2011), Reconstruction of the Infection Curve for SARS Epidemic in Beijing, China Using a Back-Projection Method, Communications in Statistics - Simulation and Computation, 37(2):425-433.

Associations of Age and Sex on Clinical Outcome and Incubation Period of Shiga toxin-producing Escherichia coli O104:H4 Infections, 2011 (2013), Werber D, King LA, Müller L, Follin P, Buchholz U, Bernard H, Rosner BM, Ethelberg S, de Valk H, Höhle M, American Journal of Epidemiology, 178(6):984-992.

Examples

```
#Generate an artificial outbreak of size n starting at time t0 and being of length
n <- 1e3 ; t0 <- 23 ; l <- 10

#PMF of the incubation time is an interval censored gamma distribution
#with mean 15 truncated at 25.
dmax <- 25
inc.pmf <- c(0,(pgamma(1:dmax,15,1.4) - pgamma(0:(dmax-1),15,1.4))/pgamma(dmax,15,1.4))
#Function to sample from the incubation time
rincu <- function(n) {
  sample(0:dmax, size=n, replace=TRUE, prob=inc.pmf)
}
#Sample time of exposure and length of incubation time
set.seed(123)
exposureTimes <- t0 + sample(x=0:(l-1),size=n,replace=TRUE)
symptomTimes <- exposureTimes + rincu(n)

#Time series of exposure (truth) and symptom onset (observed)
X <- table( factor(exposureTimes,levels=1:(max(symptomTimes)+dmax)))
Y <- table( factor(symptomTimes,levels=1:(max(symptomTimes)+dmax)))
#Convert Y to an sts object
```

```
sts <- new("sts", epoch=1:length(Y),observed=matrix(Y,ncol=1))

#Plot the outbreak
plot(sts, xaxis.labelFormat=NULL, legend=NULL)
#Add true number of exposures to the plot
lines(1:length(Y)+0.2,X,col="red",type="h",lty=2)


#Helper function to show the EM step
plotIt <- function(cur.sts) {
  plot(cur.sts,xaxis.labelFormat=NULL, legend=NULL,ylim=c(0,140))
}

#Call non-parametric back-projection function with hook function but
#without bootstrapped confidence intervals
bpnp.control <- list(k=0,eps=rep(0.005,2),iter.max=rep(250,2),B=-1,hookFun=plotIt,verbose=TRUE)

#Fast C version (use argument: eq3a.method="C")!
system.time(sts.bp <- backprojNP(sts, incu.pmf=inc.pmf,
    control=modifyList(bpnp.control,list(eq3a.method="C")), ylim=c(0,max(X,Y))))

#Show result
plot(sts.bp,xaxis.labelFormat=NULL,legend=NULL,lwd=c(1,1,2),lty=c(1,1,1),main="")
lines(1:length(Y)+0.2,X,col="red",type="h",lty=2)

#Do the convolution for the expectation
mu <- matrix(0,ncol=ncol(sts.bp),nrow=nrow(sts.bp))
#Loop over all series
for (j in 1:ncol(sts.bp)) {
  #Loop over all time points
  for (t in 1:nrow(sts.bp)) {
    #Convolution, note support of inc.pmf starts at zero (move idx by 1)
    i <- seq_len(t)
    mu[t,j] <- sum(inc.pmf[t-i+1] * upperbound(sts.bp)[i,j],na.rm=TRUE)
  }
}
#Show the fit
lines(1:nrow(sts.bp)-0.5,mu[,1],col="green",type="s",lwd=3)

#Non-parametric back-projection including boostrap CIs. B=10 is only
#used for illustration in the documentation example
#In practice use a realistic value of B=1000 or more.
bpnp.control2 <- modifyList(bpnp.control, list(hookFun=NULL,k=2,B=10,eq3a.method="C"))
## Not run:
bpnp.control2 <- modifyList(bpnp.control, list(hookFun=NULL,k=2,B=1000,eq3a.method="C"))

## End(Not run)
sts.bp2 <- backprojNP(sts, incu.pmf=inc.pmf, control=bpnp.control2)

######################################################################
# Plot the result. This is currently a manual routine.
# ToDo: Need to specify a plot method for stsBP objects which also
#       shows the CI.
```

```
#
# Parameters:
#  stsBP - object of class stsBP which is to be plotted.
######################################################################

plot.stsBP <- function(stsBP) {
  maxy <- max(observed(stsBP),upperbound(stsBP),stsBP@ci,na.rm=TRUE)
  plot(upperbound(stsBP),type="n",ylim=c(0,maxy), ylab="Cases",xlab="time")
  if (!all(is.na(stsBP@ci))) {
    polygon( c(1:nrow(stsBP),rev(1:nrow(stsBP))),
             c(stsBP@ci[2,,1],rev(stsBP@ci[1,,1])),col="lightgray")
  }
  lines(upperbound(stsBP),type="l",lwd=2)
  legend(x="topright",c(expression(lambda[t])),lty=c(1),col=c(1),fill=c(NA),border=c(NA),lwd=c(2))

  invisible()
}

#Plot the result of k=0 and add truth for comparison. No CIs available
plot.stsBP(sts.bp)
lines(1:length(Y),X,col=2,type="h")
#Same for k=2
plot.stsBP(sts.bp2)
lines(1:length(Y),X,col=2,type="h")
```

`bestCombination` *Partition of a number into two factors*

Description

Given a prime number factorization `x`, `best combination` partitions `x` into two groups, such that the product of the numbers in group one is as similar as possible to the product of the numbers of group two. This is useful in magic.dim

Usage

```
bestCombination(x)
```

Arguments

`x`	prime number factorization

Value

Returns a vector c(prod(set1),prod(set2))

boda *Surveillance for an univariate count data time series using the Bayesian Outbreak Detection Algorithm (BODA) described in Manitz and Hoehle (2013)*

Description

The function takes `range` values of the surveillance time series `sts` and for each time point uses a negative binomial regression model to compute the predictive posterior distribution for the current observation. The $(1-\alpha)\cdot 100\%$ quantile of this predictive distribution is then used as bound: If the actual observation is above the bound an alarm is raised.

Usage

```
boda(sts, control=list(range=NULL, X=NULL, trend=FALSE,
     season=FALSE, prior=c('iid','rw1','rw2'), alpha=0.05,
     mc.munu=100, mc.y=10, verbose=FALSE,multicore=TRUE))
```

Arguments

`sts`	object of class sts (including the `observed` and the `state` time series)
`control`	Control object given as a `list` containing the following components:
	`range` Specifies the index of all timepoints which should be tested. If range is `NULL` all possible timepoints are used.
	`X`
	`trend` Boolean indicating whether a linear trend term should be included in the model for the expectation the log-scale
	`season` Boolean to indicate wheather a cyclic spline should be included.
	`alpha` The threshold for declaring an observed count as an aberration is the $(1-\alpha)\cdot 100\%$ quantile of the predictive posterior.
	`mc.munu`
	`mc.y` Number of samples of y to generate for each par of the mean and size parameter. A total of $mc.munu \times mc.y$ samples are generated.
	`verbose` Argument sent to the inla call. When using ESS it might be necessary to force verbose mode for INLA to work.
	`multicore` Detect using `parallel::detectCores` how many logical cores are available and set INLA to use this number.

Details

Note: This function requires presence of the INLA R package, which is NOT available from CRAN. It can can be downloaded by calling `source("http://www.math.ntnu.no/inla/givemeINLA.R")` as described in detail at http://www.r-inla.org/download.

WARNING: This function is currently experimental!! It also heavily depends on the INLA package so changes here might affect the operational ability of the function. Since the computations for

the Bayesian GAM are quite involved do not expected this function to be particularly fast. Future work could focus on improving the speed, e.g. one issue would be to make the inference work in sequential fashion.

Author(s)

J. Manitz and M. Höhle

References

Bayesian model algorithm for monitoring reported cases of campylobacteriosis in Germany (2013), Manitz J and Höhle M, Biometrical Journal, 55(4), pp. 509 526.

Examples

```
  #Load the campylobacteriosis data for Germany
  data("campyDE")
  #Make an sts object from the data.frame
  cam.sts <-  new("sts",epoch=as.numeric(campyDE$date),observed=campyDE$case,
                  state=campyDE$state, epochAsDate=TRUE)

  ## Not run:
  #Trial run code. To be made an actual working example.
  #source("../R/boda.R")

  # define monitoring period
#  range <- which(epoch(cam.sts)>=as.Date("2007-01-01"))
#  range <- which(epoch(cam.sts)>=as.Date("2011-12-10"))
  range <- tail(1:nrow(cam.sts),n=2)

  control <- list(range=range, X=NULL, trend=TRUE, season=TRUE,
                  prior='iid', alpha=0.025, mc.munu=100, mc.y=10)

  #Apply the boda algorithm in its simples form, i.e. spline is
  #described by iid random effects and no extra covariates
  #(NOTE: requires the INLA package to be installed)
  cam.boda1 <- boda(cam.sts, control=control)

  #In case INLA is not installed, boda throws an error
  if(!inherits(cam.boda1,'try-error')){
    plot(cam.boda1,xlab='time [weeks]', ylab='No. reported',dx.upperbound=0)
  }


## End(Not run)
```

`campyDE` *Cases of Campylobacteriosis and Absolute Humidity in Germany 2002-2011*

Description

Weekly number of reported campylobacteriosis cases in Germany 2002-2011 together with the corresponding absolute humidity (in g/m^3) that week. The absolute humidity was computed according to the procedure by Dengler (1997) using the means of representative weather station data from the German Climate service.

Usage

```
data(campyDE)
```

Format

A `data.frame` containing the following columns

`date` `Date` instance containing the monday of the reporting week.

`case` Number of reported cases that week.

`state` Boolean indicating whether there is external knowledge about an outbreak that week

`hum` Mean absolute humidity (in g/m^3) of that week as measured by a single representative weather station.

`l1.hum`-`l5.hum` Lagged version (lagged by 1-5) of the `hum` covariate.

newyears Boolean indicating whether the reporting week corresponds to the first two weeks of the year (TRUE) or not (FALSE). Note: The first week of a year is here defined as the first reporting week, which has its corresponding monday within new year.

christmas Boolean indicating whether the reporting week corresponds to the last two weeks of the year (TRUE) or not (FALSE). Note: This are the first two weeks before the `newyears` weeks.

O104period Boolean indicating whether the reporting week corresponds to the W21-W30 period of increased gastroenteritis awareness during the O104:H4 STEC outbreak.

Source

The data on campylobacteriosis cases are queried from the Survstat@RKI database of the German Robert Koch Institute (http://www3.rki.de/SurvStat/).

Data for the computation of absolute humidity were obtained from the German Climiate Service (Deutscher Wetterdienst), Climate data of Germany, available at http://www.dwd.de.

A complete data description and an analysis of the data can be found in:

Bayesian model algorithm for monitoring reported cases of campylobacteriosis in Germany (2013), Manitz J and Höhle M, Biometrical Journal, 55(4), pp. 509 526.

Examples

```
#Load the data
data("campyDE")

#O104 period is W21-W30 in 2011
stopifnot(all(campyDE$O104period == (
    (campyDE$date >= as.Date("2011-05-23")) &
    (campyDE$date < as.Date("2011-07-31"))
```

```
)))

#Make an sts object from the data.frame
cam.sts <-  new("sts",epoch=as.numeric(campyDE$date),observed=campyDE$case,
                state=campyDE$state, epochAsDate=TRUE)

#Plot the result
plot(cam.sts)
```

`categoricalCUSUM` *CUSUM detector for time-varying categorical time series*

Description

Function to process `sts` object by binomial, beta-binomial or multinomial CUSUM. Logistic, multinomial logistic, proportional odds or Bradley-Terry regression models are used to specify in-control and out-of-control parameters.

Usage

```
categoricalCUSUM(stsObj,control = list(range=NULL,h=5,pi0=NULL,
                 pi1=NULL, dfun=NULL, ret=c("cases","value")),...)
```

Arguments

`stsObj`	Object of class `sts` containing the number of counts in each of the k categories of the response variable. Time varying number of counts n_t is found in slot `populationFrac`.
`control`	Control object containing several items • `range`Vector of length t_{max} with indices of the `observed` slot to monitor. • `h`Threshold to use for the monitoring. Once the CUSUM statistics is larger or equal to `h` we have an alarm. • `pi0`$(k-1)\times t_{max}$ in-control probability vector for all categories except the reference category. • `mu1`$(k-1)\times t_{max}$ out-of-control probability vector for all categories except the reference category. • `dfun`The probability mass function or density used to compute the likelihood ratios of the CUSUM. In a negative binomial CUSUM this is `dnbinom`, in a binomial CUSUM `dbinom` and in a multinomial CUSUM `dmultinom`. The function must be able to handle the arguments `y`, `size`, `mu` and `log`. As a consequence, one in the case of the beta-binomial distribution has to write a small wrapper function. • `ret`Return the necessary proportion to sound an alarm in the slot `upperbound` or just the value of the CUSUM statistic. Thus, `ret` is one of tha values in `c("cases","value")`.
`...`	Additional arguments to send to `dfun`.

Details

The function allows the monitoring of categorical time series as described by regression models for binomial, beta-binomial or multinomial data. The later includes e.g. multinomial logistic regression models, proportional odds models or Bradley-Terry models for paired comparisons. See the Höhle (2010) reference for further details about the methodology.

Once an alarm is found the CUSUM scheme is resetted (to zero) and monitoring continues from there.

Value

An `sts` object with `observed`, `alarm`, etc. slots trimmed to the `control$range` indices.

Author(s)

M. Höhle

References

Höhle, M. (2010), Changepoint detection in categorical time series, Book chapter to appear in T. Kneib and G. Tutz (Eds.), Statistical Modelling and Regression Structures, Springer.

See Also

categoricalCUSUM

Examples

```
if (require("gamlss")) {
  ###########################################################################
  #Beta-binomial CUSUM for a small example containing the time-varying
  #number of positive test out of a time-varying number of total
  #test.
  ######################################

  #Load meat inspection data
  data("abattoir")

  #Use GAMLSS to fit beta-bin regression model
  phase1 <- 1:(2*52)
  phase2  <- (max(phase1)+1) : nrow(abattoir)

  #Fit beta-binomial model using GAMLSS
  abattoir.df <- as.data.frame(abattoir)
  colnames(abattoir.df) <- c("y","t","state","alarm","n")
  m.bbin <- gamlss( cbind(y,n-y) ~ 1 + t +
    + sin(2*pi/52*t) + cos(2*pi/52*t) +
    + sin(4*pi/52*t) + cos(4*pi/52*t), sigma.formula=~1,
    family=BB(sigma.link="log"),
    data=abattoir.df[phase1,c("n","y","t")])

  #CUSUM parameters
```

```
  R <- 2 #detect a doubling of the odds for a test being positive
  h <- 4 #threshold of the cusum

  #Compute in-control and out of control mean
  pi0 <- predict(m.bbin,newdata=abattoir.df[phase2,c("n","y","t")],type="response")
  pi1 <- plogis(qlogis(pi0)+log(R))
  #Create matrix with in control and out of control proportions.
  #Categories are D=1 and D=0, where the latter is the reference category
  pi0m <- rbind(pi0, 1-pi0)
  pi1m <- rbind(pi1, 1-pi1)


  ######################################################################
  # Use the multinomial surveillance function. To this end it is necessary
  # to create a new abattoir object containing counts and proportion for
  # each of the k=2 categories. For binomial data this appears a bit
  # redundant, but generalizes easier to k>2 categories.
  ######################################################################

  abattoir2 <- new("sts",epoch=1:nrow(abattoir), start=c(2006,1),freq=52,
    observed=cbind(abattoir@observed,abattoir@populationFrac -abattoir@observed),
    populationFrac=cbind(abattoir@populationFrac,abattoir@populationFrac),
    state=matrix(0,nrow=nrow(abattoir),ncol=2),
    multinomialTS=TRUE)

  ######################################################################
  #Function to use as dfun in the categoricalCUSUM
  #(just a wrapper to the dBB function). Note that from v 3.0-1 the
  #first argument of dBB changed its name from "y" to "x"!
  ######################################################################
  mydBB.cusum <- function(y, mu, sigma, size, log = FALSE) {
    return(dBB(y[1,], mu = mu[1,], sigma = sigma, bd = size, log = log))
  }


  #Create control object for multinom cusum and use the categoricalCUSUM
  #method
  control <- list(range=phase2,h=h,pi0=pi0m, pi1=pi1m, ret="cases",
   dfun=mydBB.cusum)
  surv <- categoricalCUSUM(abattoir2, control=control,
   sigma=exp(m.bbin$sigma.coef))

  #Show results
  plot(surv[,1],legend.opts=NULL,dx.upperbound=0)
  lines(pi0,col="green")
  lines(pi1,col="red")

  #Index of the alarm
  which.max(alarms(surv[,1]))
}

#ToDo: Compute run length using LRCUSUM.runlength
```

checkResidualProcess *Check the residual process of a fitted* twinSIR *or* twinstim

Description

Transform the residual process (cf. the residuals methods for classes "twinSIR" and "twinstim") such that the transformed residuals should be uniformly distributed if the fitted model well describes the true conditional intensity function. Graphically check this using ks.plot.unif. The transformation for the residuals tau is 1 - exp(-diff(c(0,tau))) (cf. Ogata, 1988). Another plot inspects the serial correlation between the transformed residuals (scatterplot between u_i and u_i+1).

Usage

```
checkResidualProcess(object, plot = 1:2, mfrow = c(1,length(plot)), ...)
```

Arguments

object	an object of class "twinSIR" or "twinstim".
plot	logical (or integer index) vector indicating if (which) plots of the transformed residuals should be produced. The plot index 1 corresponds to a ks.plot.unif to check for deviations of the transformed residuals from the uniform distribution. The plot index 2 corresponds to a scatterplot of u_i vs. u_{i+1}. By default (plot = 1:2), both plots are produced.
mfrow	see par.
...	further arguments passed to ks.plot.unif.

Value

A list (returned invisibly, if plot = TRUE) with the following components:

tau the residual process obtained by residuals(object).

U the transformed residuals which should be distributed as U(0,1).

ks the result of the ks.test for the uniform distribution of U.

Author(s)

Sebastian Meyer

References

Ogata, Y. (1988) Statistical models for earthquake occurrences and residual analysis for point processes. *Journal of the American Statistical Association*, 83, 9-27

See Also

ks.plot.unif and the residuals-method for classes "twinSIR" and "twinstim".

Examples

```
## load the twinSIR() fit
data("foofit")
checkResidualProcess(foofit)
```

`compMatrix.writeTable` *Latex Table Generation*

Description

generates a latex table

Usage

```
compMatrix.writeTable(compMatrix)
```

Arguments

`compMatrix`	Matrix which includes quality values for every surveillance system.

Value

`xtable`	Latex table of the entered matrix.

Author(s)

M. Höhle, A. Riebler, C. Lang

Examples

```
### First creates some tables ###

# Create a test object
disProgObj1 <- sim.pointSource(p = 0.99, r = 0.5, length = 400,
                               A = 1, alpha = 1, beta = 0, phi = 0,
                               frequency = 1, state = NULL, K = 1.7)
disProgObj2 <- sim.pointSource(p = 0.99, r = 0.5, length = 400,
                               A = 1, alpha = 1, beta = 0, phi = 0,
                               frequency = 1, state = NULL, K = 5)
disProgObj3 <- sim.pointSource(p = 0.99, r = 0.5, length = 400,
                               A = 1, alpha = 1, beta = 0, phi = 0,
                               frequency = 1, state = NULL, K = 17)

# Let this object be tested from any methods in range = 200:400
range <- 200:400
control <- list( list(funcName = "rki1", range = range),
                 list(funcName = "rki2", range = range),
                 list(funcName = "rki3", range = range)
           )
```

```
### This are single compMatrices
compMatrix1 <- algo.compare(algo.call(disProgObj1, control=control))
compMatrix2 <- algo.compare(algo.call(disProgObj2, control=control))
compMatrix3 <- algo.compare(algo.call(disProgObj3, control=control))

### This is a summary compMatrix
sumCompMatrix <- algo.summary( list(a=compMatrix1,
                                    b=compMatrix2, c=compMatrix3) )

### Now show the latextable from the single compMatrix compMatrix1
compMatrix.writeTable(compMatrix1)

### Now show the latextable from the summary compMatrix
compMatrix.writeTable(sumCompMatrix)
```

`correct53to52` *Data Correction from 53 to 52 weeks*

Description

Correction of data from 53 to 52 weeks a year

Usage

```
correct53to52(disProgObj, firstweek = 1)
```

Arguments

`disProgObj`	object of class disProg (including the observed and the state chain).
`firstweek`	the number of the first week in a year, default = 1 (if it starts with the beginning of a year). Necessary, because the infected of week 53 and the infected of week 52 must be added.

Details

`readData` reads data with 53 weeks a year, but normally one year is said to have 52 weeks.

Value

`disProg`	a object `disProg` (disease progress) including a list of the observed and the state chain (corrected to 52 weeks instead of 53 weeks a year)

See Also

`readData`

Examples

```
#This call correct53to52 automatically
obj <- readData("k1",week53to52=TRUE)
correct53to52(obj) # first entry is the first week of the year

obj <- readData("n1",week53to52=FALSE)
correct53to52(obj, firstweek = 5) # now it's assumed that the fifth
                                  # entry is the first week of the year
```

`create.disProg` *Creating an object of class disProg*

Description

Creates an object of class `disProg` from a vector with the weeknumber (week) and matrices with the observed number of counts (observed) and the respective state chains (state), where each column represents an individual time series. The matrices neighbourhood and populationFrac provide information about neighbouring units and population proportions.

Usage

```
create.disProg(week, observed, state, start=c(2001,1), freq=52,
neighbourhood=NULL, populationFrac=NULL, epochAsDate=FALSE)
```

Arguments

`week`	index in the matrix of observations, typically weeks
`observed`	matrix with parallel time series of counts where rows are time points and columns are the individual time series for unit/area $i, i = 1, \ldots, m$
`state`	matrix with corresponding states
`start`	vector of length two denoting the year and the sample number (week, month, etc.) of the first observation
`freq`	sampling frequency per year, i.e. 52 for weekly data, 12 for monthly data, 13 if 52 weeks are aggregated into 4 week blocks.
`neighbourhood`	neighbourhood matrix N of dimension $m \times m$ with elements $n_{ij} = 1$ if units i and j are adjacent and 0 otherwise
`populationFrac`	matrix with corresponding population proportions
`epochAsDate`	interpret the integers in `week` as Dates. Default is `FALSE`

Value

`disProg`	object of class `disProg`

Author(s)

M. Paul

Examples

```
# create an univariate disProg object
# read in salmonella.agona data
salmonella <- read.table(system.file("extdata/salmonella.agona.txt",
                                     package = "surveillance"), header = TRUE)
# look at data.frame
str(salmonella)

salmonellaDisProg <- create.disProg(week = 1:nrow(salmonella),
      observed = salmonella$observed,
      state = salmonella$state, start = c(1990, 1))

# look at disProg object
salmonellaDisProg
```

`create.grid` *Computes a matrix of initial values*

Description

For a given model and a list of parameters specified as `param = c(lower,upper,length)`, `create.grid` creates a grid of initial values for `algo.hhh.grid`. The resulting matrix contains all combinations of the supplied parameters which each are a sequence of length `length` from `lower` to `upper`. Note that the autoregressive parameters λ, ϕ and the overdispersion parameter ψ must be positive. Only one sequence of initial values is considered for the autregressive, endemic and overdispersion parameters to create the grid, e.g. initial values are the same for each one of the seasonal and trend parameters.

Usage

```
create.grid(disProgObj, control, params = list(epidemic = c(0.1, 0.9, 5),
   endemic=c(-0.5,0.5,3), negbin = c(0.3, 12, 10)))
```

Arguments

`disProgObj`	object of class `disProg`
`control`	specified model
`params`	list of parameters: `param=c(lower,upper,length)` • `epidemic` autoregressive parameters λ and ϕ. • `endemic` trend and seasonal parameters β, γ_j. • `negbin` overdispersion parameter for negative binomial model ψ.

Value

`matrix`	matrix with `gridSize` starting values as rows

Author(s)

M. Paul

See Also

algo.hhh.grid

Examples

```
# simulate data
set.seed(123)
disProgObj <- simHHH(control = list(coefs = list(alpha =-0.5, gamma = 0.4,
                    delta = 0.6)),length=300)$data

# consider the model specified in a control object for algo.hhh.grid
cntrl1 <- list(lambda=TRUE, neighbours=TRUE,
               linear=TRUE, nseason=1)
cntrl2 <- list(lambda=TRUE, negbin="single")

# create a grid of initial values for respective parameters
grid1 <- create.grid(disProgObj, cntrl1,
                    params = list(epidemic=c(0.1,0.9,3),
                                  endemic=c(-1,1,3)))
grid2 <- create.grid(disProgObj, cntrl2,
                    params = list(epidemic=c(0.1,0.9,5),
                                  negbin=c(0.3,12,10)))
```

---

deleval *Surgical failures data*

Description

The dataset from Steiner et al. (1999) on A synthetic dataset from the Danish meat inspection – useful for illustrating the beta-binomial CUSUM.

Usage

```
data(abattoir)
```

Details

Steiner et al. (1999) use data from de Leval et al. (1994) to illustrate monitoring of failure rates of a surgical procedure for a bivariate outcome.

Over a period of six years an arterial switch operation was performed on 104 newborn babies. Since the death rate from this surgery was relatively low the idea of surgical "near miss" was introduced. It is defined as the need to reinstitute cardiopulmonary bypass after a trial period of weaning. The

object of class `sts` contains the recordings of near misses and deaths from the surgery for the 104 newborn babies of the study.

The data could also be handled by a multinomial CUSUM model.

References

Steiner, S. H., Cook, R. J., and Farewell, V. T. (1999), Monitoring paired binary surgical outcomes using cumulative sum charts, Statistics in Medicine, 18, pp. 69–86.

De Leval, Marc R., Franiois, K., Bull, C., Brawn, W. B. and Spiegelhalter, D. (1994), Analysis of a cluster of surgical failures, Journal of Thoracic and Cardiovascular Surgery, March, pp. 914–924.

See Also

`pairedbinCUSUM`

Examples

```
data("deleval")
plot(deleval, xaxis.labelFormat=NULL,ylab="Response",xlab="Patient number")
```

`discpoly` *Polygonal Approximation of a Disc/Circle*

Description

Generates a polygon representing a disc/circle (in planar coordinates) as an object of one of three possible classes: `"Polygon"`, `"owin"`, or – if **rgeos** (or **gpclib**) are available – `"gpc.poly"`.

Usage

```
discpoly(center, radius, npoly = 64,
         class = c("Polygon", "owin", "gpc.poly"),
         hole = FALSE)
```

Arguments

`center`	numeric vector of length 2 (center coordinates of the circle).
`radius`	single numeric value (radius of the circle).
`npoly`	single integer. Number of edges of the polygonal approximation.
`class`	class of the resulting polygon (partial name matching applies). For `"owin"`, this is just a wrapper around **spatstat**'s own `disc` function.
`hole`	logical. Does the resulting polygon represent a hole?

Value

A polygon of class `class` representing a circle/disc with `npoly` edges accuracy.
If `class="gpc.poly"` although this formal class is not currently defined (and **rgeos** is not available), only the `pts` slot of a `"gpc.poly"` is returned with a warning.

Author(s)

Sebastian Meyer
This function is inspired by the `disc` function from package **spatstat** authored by Adrian Baddeley and Rolf Turner.

See Also

disc in package **spatstat**.

Examples

```
## Construct circles with increasing accuracy and of different spatial classes
disc1 <- discpoly(c(0,0), 5, npoly=4, class = "owin")
disc2 <- discpoly(c(0,0), 5, npoly=16, class = "Polygon")

## Look at the results
print(disc1)
plot(disc1, axes=TRUE, main="", border=2)

print(disc2)
lines(disc2, col=3)

if (requireNamespace("rgeos")) { # for the "gpc.poly" class definition
    disc3 <- discpoly(c(0,0), 5, npoly=64, class = "gpc.poly")
    print(disc3)
    plot(disc3, add=TRUE, poly.args=list(border=4))
}

## if one only wants to _draw_ a circle without an object behind
symbols(0, 0, circles=5, inches=FALSE, add=TRUE, fg=5)
```

---

disProg2sts *Convert disProg object to sts and vice versa*

Description

A small helper function to convert a `disProg` object to become an object of the S4 class `sts` and vice versa. In the future the `sts` should replace the `disProg` class, but for now this function allows for conversion between the two formats.

Usage

```
disProg2sts(disProgObj, map=NULL)
sts2disProg(sts)
```

Arguments

disProgObj	object of class disProg
map	SpatialPolygonsDataFrame object containing the map visualization
sts	Object of class sts to convert

Value

an object of class sts or disProg, respectively.

See Also

sts-class

Examples

```
data(ha)
print(disProg2sts(ha))
class(sts2disProg(disProg2sts(ha)))
```

---

earsC — *Surveillance for a count data time series using the EARS C1, C2 or C3 method.*

Description

The function takes range values of the surveillance time series sts and for each time point computes a threshold for the number of counts based on values from the recent past. This is then compared to the observed number of counts. If the observation is above a specific quantile of the prediction interval, then an alarm is raised. This method is especially useful for data without many reference values, since it only needs counts from the recent past.

Usage

```
earsC(sts, control = list(range = NULL, method = "C1",
                          alpha = 0.001))
```

Arguments

sts	object of class sts (including the observed and the state time series) , which is to be monitored.
control	Control object
	range Specifies the index of all timepoints which should be tested. If range is NULL the maximum number of possible timepoints is used. This number depends on the method chosen. For C1 all timepoints from timepoint 8 can be assessed, for C2 from timepoint 10 and for C3 from timepoint 12.

`method` String indicating which method to use:
"C1" for EARS C1-MILD method, "C2" for EARS C2-MEDIUM method, "C3" for EARS C3-HIGH method. By default if `method` is `NULL` C1 is chosen.

`alpha` An approximate (two-sided) $(1-\alpha)\cdot 100\%$ prediction interval is calculated. By default if `alpha` is `NULL` 0.001 is assumed for C1 and C2 whereas 0.025 is assumed for C3. These different choices are the one made at the CDC.

Details

The three methods are different in terms of baseline used for calculation of the expected value and in terms of method for calculating the expected value:

- in C1 and C2 the expected value is the moving average of counts over the sliding window of the baseline and the prediction interval depends on the standard derivation of counts over this window. They can be considered as Shewhart control charts with a small sample used for calculations.
- in C3 the expected value is based on the sum over 3 timepoints (assessed timepoints and the two previous timepoints) of the discrepancy between observations and predictions, predictions being calculated with C2 method. This method shares a common point with CUSUM method (adding discrepancies between predictions and observations over several timepoints) but is not a CUSUM (sum over 3 timepoints, not accumulation over a whole range), even if it sometimes presented as such.

Here is what the function does for each method:

1. For C1 the baseline are the 7 timepoints before the assessed timepoint t, t-7 to t-1. The expected value is the mean of the baseline. An approximate (two-sided) $(1-\alpha)\cdot 100\%$ prediction interval is calculated based on the assumption that the difference between the expected value and the observed value divided by the standard derivation of counts over the sliding window, called $C_1(t)$, follows a standard normal distribution in the absence of outbreaks:

$$C_1(t) = \frac{Y(t) - \bar{Y}_1(t)}{S_1(t)},$$

where

$$\bar{Y}_1(t) = \frac{1}{7}\sum_{i=t-1}^{t-7} Y(i)$$

and

$$S_1^2(t) = \frac{1}{6}\sum_{i=t-1}^{t-7} [Y(i) - \bar{Y}_1(i)]^2.$$

Then under the null hypothesis of no outbreak,

$$C_1(t) \sim N(0,1)$$

An alarm is raised if

$$C_1(t) \geq z_{1-\alpha}$$

with $z_{1-\alpha}$ the $(1-\alpha)^{th}$ quantile of the centered reduced normal law.

The upperbound $U_1(t)$ is then defined by:

$$U_1(t) = \bar{Y}_1(t) + z_{1-\alpha}S_1(t).$$

2. C2 is very close to C1 apart from a 2-day lag in the baseline definition. Indeed for C2 the baseline are 7 timepoints with a 2-day lag before the assessed timepoint t, t-9 to t-3. The expected value is the mean of the baseline. An approximate (two-sided) $(1-\alpha)\cdot 100\%$ prediction interval is calculated based on the assumption that the difference between the expected value and the observed value divided by the standard derivation of counts over the sliding window, called $C_2(t)$, follows a standard normal distribution in the absence of outbreaks:

$$C_2(t) = \frac{Y(t) - \bar{Y}_2(t)}{S_2(t)},$$

where

$$\bar{Y}_2(t) = \frac{1}{7}\sum_{i=t-3}^{t-9} Y(i)$$

and

$$S_2^2(t) = \frac{1}{6}\sum_{i=t-3}^{t-9} [Y(i) - \bar{Y}_2(i)]^2.$$

Then under the null hypothesis of no outbreak,

$$C_2(t) \sim N(0,1)$$

An alarm is raised if

$$C_2(t) \geq z_{1-\alpha},$$

with $z_{1-\alpha}$ the $(1-\alpha)^{th}$ quantile of the centered reduced normal law.

The upperbound $U_2(t)$ is then defined by:

$$U_2(t) = \bar{Y}_2(t) + z_{1-\alpha}S_2(t).$$

3. C3 is quite different from the two other methods but it is based on C2. Indeed it uses $C_2(t)$ from timepoint t and the two previous timepoints. This means the baseline are timepoints t-11 to t-3. The statistic $C_3(t)$ is the sum of discrepancies between observations and predictions.

$$C_3(t) = \sum_{i=t}^{t-2} \max(0, C_2(i) - 1)$$

Then under the null hypothesis of no outbreak,

$$C_3(t) \sim N(0,1)$$

An alarm is raised if

$$C_3(t) \geq z_{1-\alpha},$$

with $z_{1-\alpha}$ the $(1-\alpha)^{th}$ quantile of the centered reduced normal law.

The upperbound $U_3(t)$ is then defined by:

$$U_3(t) = \bar{Y}_2(t) + S_2(t)\left(z_{1-\alpha} - \sum_{i=t-1}^{t-2} \max(0, C_2(i) - 1)\right).$$

Value

An object of class `sts` with the slots `upperbound` and `alarm` filled by the chosen method.

Author(s)

M. Salmon

Source

Fricker, R.D., Hegler, B.L, and Dunfee, D.A. (2008). Comparing syndromic surveillance detection methods: EARS versus a CUSUM-based methodology, 27:3407-3429, Statistics in medicine.

Examples

```
#Sim data and convert to sts object
disProgObj <- sim.pointSource(p = 0.99, r = 0.5, length = 208, A = 1,
                              alpha = 1, beta = 0, phi = 0,
                              frequency = 1, state = NULL, K = 1.7)
stsObj = disProg2sts( disProgObj)


#Call function and show result
res1 <- earsC(stsObj, control = list(range = 20:208,method="C1"))
plot(res1,legend.opts=list(horiz=TRUE,x="topright"),dx.upperbound=0)


# compare upperbounds depending on alpha
res3 <- earsC(stsObj, control = list(range = 20:208,method="C3",alpha = 0.001))
plot(res3@upperbound,t='l')
res3 <- earsC(stsObj, control = list(range = 20:208,method="C3"))
lines(res3@upperbound,col='red')
```

enlargeData *Data Enlargement*

Description

Enlargement of data which is too short for a surveillance method to evaluate.

Usage

```
enlargeData(disProgObj, range = 1:156, times = 1)
```

Arguments

disProgObj	object of class disProg (including the observed and the state chain).
range	range of already existing data (state, observed) which should be used for enlargement.
times	number of times to enlarge.

Details

observed and state are enlarged in the way that the part range of observed and state is repeated times times in front of observed and state. Sometimes it's useful to care for the cyclic property of the timeseries, so as default we enlarge observed and state once with the first three existing years, assuming a year has 52 weeks.

Value

disProg	a object disProg (disease progress) including a list of the observed and the state chain (extended with cyclic data generation)

See Also

readData

Examples

```
obj <- readData("k1")

enlargeData(obj) # enlarge once with part 1:156
enlargeData(obj, 33:36, 10) # enlarge 10 times with part 33:36
```

epidata *Continuous-Time SIR Event History of a Fixed Population*

Description

The function `as.epidata` is used to generate objects of `class` `"epidata"`. Objects of this class are specific data frames containing the event history of an epidemic together with some additional attributes. These objects are the basis for fitting spatio-temporal epidemic intensity models with the function `twinSIR`. Note that the spatial information itself, i.e. the positions of the individuals, is assumed to be constant over time. Besides epidemics following the SIR compartmental model, also data from SI, SIRS and SIS epidemics may be supplied. Inference for the infectious process works as usual and simulation of such epidemics is also possible.

Usage

```
as.epidata(data, ...)

## S3 method for class 'data.frame'
as.epidata(data, t0, tE.col, tI.col, tR.col,
           id.col, coords.cols, f = list(), w = list(), keep.cols = TRUE, ...)
## Default S3 method:
as.epidata(data, id.col, start.col, stop.col, atRiskY.col,
           event.col, Revent.col, coords.cols, f = list(), w = list(), ...)

## S3 method for class 'epidata'
print(x, ...)
## S3 method for class 'epidata'
x[i, j, drop]
## S3 method for class 'epidata'
update(object, f = list(), w = list(), ...)
```

Arguments

`data`	For the `data.frame`-method, a data frame with as many rows as there are individuals in the population and time columns indicating when each individual became exposed (optional), infectious (mandatory, but can be `NA` for non-affected individuals) and removed (optional). Note that this data format does not allow for re-infection (SIRS) and time-varying covariates. The `data.frame`-method converts the individual-indexed data frame to the long event history start/stop format and then feeds it into the default method. If calling the generic function `as.epidata` on a `data.frame` and the `t0` argument is missing, the default method is called directly. For the default method, `data` can be a `matrix` or a `data.frame`. It must contain the observed event history in a form similar to `Surv(, type="counting")` with additional information (variables) along the process. Rows will be sorted automatically during conversion. The observation period is splitted up into *consecutive* intervals of constant state - thus constant infection intensities. The data

frame consists of a block of N (number of individuals) rows for each of those time intervals (all rows in a block have the same start and stop values... therefore the name "block"), where there is one row per individual in the block. Each row describes the (fixed) state of the individual during the interval given by the start and stop columns `start.col` and `stop.col`.
Note that there may not be more than one event (infection or removal) in a single block. Thus, in a single block, only one entry in the `event.col` and `Revent.col` may be 1, all others are 0. This rule follows the point process characteristic that there are no concurrent events (infections or removals).

`t0` start time of the observation period. Will be subtracted from the time columns `tE.col`, `tI.col`, `tR.col`. Individuals that have already been removed prior to `t0`, i.e., rows with `tR <= t0`, will be dropped.

`tE.col, tI.col, tR.col`
single numeric or character indexes of the time columns in `data`, which specify when the individuals became exposed, infectious and removed, respectively. `tE.col` and `tR.col` can be missing, corresponding to SIR, SEI, or SI data. `NA` entries mean that the respective event has not (yet) occurred. Note that `is.na(tE)` implies `is.na(tI)` and `is.na(tR)`, and `is.na(tI)` implies `is.na(tR)` (and this is checked for the provided data).

`id.col` single numeric or character index of the `id` column in `data`. The `id` column identifies the individuals in the data frame. It is converted to a factor by calling `factor`, i.e., unused levels are dropped if it already was a factor.

`start.col` single index of the `start` column in `data`. Can be numeric (by column number) or character (by column name). The `start` column contains the (numeric) time points of the beginnings of the consecutive time intervals of the event history. The minimum value in this column, i.e. the start of the observation period should be 0.

`stop.col` single index of the `stop` column in `data`. Can be numeric (by column number) or character (by column name). The `stop` column contains the (numeric) time points of the ends of the consecutive time intervals of the event history. The stop value must always be greater than the start value of a row.

`atRiskY.col` single index of the `atRiskY` column in `data`. Can be numeric (by column number) or character (by column name). The `atRiskY` column indicates if the individual was "at-risk" of becoming infected during the time interval (start; stop]. This variable must be logical or in 0/1-coding. Individuals with `atRiskY == 0` in the first time interval (normally the rows with `start == 0`) are taken as *initially infectious*.

`event.col` single index of the `event` column in `data`. Can be numeric (by column number) or character (by column name). The `event` column indicates if the individual became *infected* at the `stop` time of the interval. This variable must be logical or in 0/1-coding.

`Revent.col` single index of the `Revent` column in `data`. Can be numeric (by column number) or character (by column name). The `Revent` column indicates if the individual was *recovered* at the `stop` time of the interval. This variable must be logical or in 0/1-coding.

`coords.cols` index*es* of the `coords` column*s* in `data`. Can be a numeric (by column number) vector, a character (by column name) vector or `NULL` (in which case epidemic

covariates are not calculateable). These columns contain the coordinates of the individuals. Note that the functions related to twinSIR currently assume *fixed positions* of the individuals during the whole epidemic. Thus, an individual has the same coordinates in every block. For simplicity, the coordinates are derived from the first time block only (normally the rows with `start == 0`). The epidemic covariates are calculated based on the Euclidian distances between the individuals, see `f`.

`f` — a *named* list of *vectorized* functions for a distance-based force of infection. The functions must interact elementwise on a (distance) matrix so that - for a matrix D - f[[m]](D) results in a matrix. A simple example is `function(u) {u <= 1}`, which indicates if the Euclidian distance between the individuals is smaller than or equal to 1. The names of the functions will be the names of the epidemic variables in the resulting data frame. So, the names should not coincide with names of other covariates. The distance-based weights are computed as follows: $I(t)$ denotes the set of infectious individuals just before time t and s_i the coordinate vector of individual i. For individual i at time t the m'th covariate has the value $\sum_{j \in I(t)} f_m(||s_i - s_j||)$

`w` — a *named* list of *vectorized* functions for extra covariate-based weights w_{ij} in the epidemic component. Each function operates on a single time-constant covariate in `data`, which is determined by the name of the first argument: The two function arguments should be named `varname.i` and `varname.j`, where `varname` is one of `names(data)`. Similar to the components in `f`, `length(w)` epidemic covariates will be generated in the resulting `"epidata"` named according to `names(w)`. So, the names should not coincide with names of other covariates. For individual i at time t, the m'th such covariate has the value $\sum_{j \in I(t)} w_m(z_i^{(m)}, z_j^{(m)})$, where $z^{(m)}$ denotes the variable in `data` associated with `w[[m]]`.

`keep.cols` — logical indicating if all columns in `data` should be retained (and not only the obligatory `"epidata"` columns), in particular any additional columns with time-constant individual-specific covariates. Alternatively, `keep.cols` can be a numeric or character vector indexing columns of `data` to keep.

`x,object` — an object of class `"epidata"`.

`...` — arguments passed to `print.data.frame`. Currently unused in the `as.epidata`-methods.

`i,j,drop` — arguments passed to `[.data.frame`.

Details

The `print` method for objects of class `"epidata"` simply prints the data frame with a small header containing the time range of the observed epidemic and the number of infected individuals. Usually, the data frames are quite long, so the summary method `summary.epidata` might be useful. Also, indexing/subsetting `"epidata"` works exactly as for `data.frame`s, but there is an own method, which assures consistency of the resulting `"epidata"` or drops this class, if necessary. The `update`-method can be used to add or replace distance-based (`f`) or covariate-based (`w`) epidemic variables in an existing `"epidata"` object.

SIS epidemics are implemented as SIRS epidemics where the length of the removal period equals 0. This means that an individual, which has an R-event will be at risk immediately afterwards, i.e.

in the following time block. Therefore, data of SIS epidemics have to be provided in that form containing "pseudo-R-events".

Value

a `data.frame` with the columns `"BLOCK"`, `"id"`, `"start"`, `"stop"`, `"atRiskY"`, `"event"`, `"Revent"` and the coordinate columns (with the original names from `data`), which are all obligatory. These columns are followed by any remaining columns of the input `data`. Last but not least, the newly generated columns with epidemic variables corresponding to the functions in the list `f` are appended, if `length(f)` > 0.

The `data.frame` is given the additional *attributes*

`"eventTimes"`	numeric vector of infection time points (sorted chronologically).
`"timeRange"`	numeric vector of length 2: `c(min(start), max(stop))`.
`"coords.cols"`	numeric vector containing the column indices of the coordinate columns in the resulting data frame.
`"f"`	this equals the argument `f`.
`"w"`	this equals the argument `w`.

Note

The column name `"BLOCK"` is a reserved name. This column will be added automatically at conversion and the resulting data frame will be sorted by this column and by id. Also the names `"id"`, `"start"`, `"stop"`, `"atRiskY"`, `"event"` and `"Revent"` are reserved for the respective columns only.

Author(s)

Sebastian Meyer

See Also

The `hagelloch` data for a "real" `"epidata"` object. The code for the conversion from the simple data frame to the SIR event history using `as.epidata.data.frame` is given in `example(hagelloch)`.

The `plot` and the `summary` method for class `"epidata"`. Furthermore, the function `animate.epidata` for the animation of epidemics.

Function `twinSIR` for fitting spatio-temporal epidemic intensity models to epidemic data.

Function `simEpidata` for the simulation of epidemic data.

Examples

```
# see example(hagelloch)

# here is an artificial event history
data("foodata")
str(foodata)

# convert the data to an object of class "epidata",
# also generating some epidemic covariates
```

```
myEpidata <- as.epidata(foodata,
  id.col = 1, start.col = "start", stop.col = "stop",
  atRiskY.col = "atrisk", event.col = "infected", Revent.col = "removed",
  coords.cols = c("x","y"),
  f = list(B1 = function(u) u <= 1, B2 = function(u) u > 1))

# this is how data("fooepidata") has been generated
data("fooepidata")
stopifnot(all.equal(myEpidata, fooepidata))

# add covariate-based weight for the force of infection, e.g.,
# to model an increased force if i and j have the same value in z1
myEpidata2 <- update(fooepidata,
                     w = list(samez1 = function(z1.i, z1.j) z1.i == z1.j))

str(fooepidata)
subset(fooepidata, BLOCK == 1)

summary(fooepidata)          # see 'summary.epidata'
plot(fooepidata)             # see 'plot.epidata' and also 'animate.epidata'
stateplot(fooepidata, "15")  # see 'stateplot'
```

---

epidataCS *Continuous Space-Time Marked Point Patterns with Grid-Based Covariates*

Description

Data structure for **c**ontinuous **s**patio-temporal event data, e.g. individual case reports of an infectious disease. Apart from the actual `events`, the class simultaneously holds a spatio-temporal grid of endemic covariates (similar to disease mapping) and a representation of the observation region.

The `"epidataCS"` class is the basis for fitting spatio-temporal epidemic intensity models with the function `twinstim`.

Usage

```
as.epidataCS(events, stgrid, W, qmatrix = diag(nTypes),
             nCircle2Poly = 32L, T = NULL,
             clipper = c("polyclip", "rgeos"), verbose = interactive())

## S3 method for class 'epidataCS'
print(x, n = 6L, digits = getOption("digits"), ...)

## S3 method for class 'epidataCS'
nobs(object, ...)
## S3 method for class 'epidataCS'
head(x, n = 6L, ...)
## S3 method for class 'epidataCS'
```

```
tail(x, n = 6L, ...)
## S3 method for class 'epidataCS'
x[i, j, ..., drop = TRUE]
## S3 method for class 'epidataCS'
subset(x, subset, select, drop = TRUE, ...)

## S3 method for class 'epidataCS'
marks(x, coords = TRUE, ...)

## S3 method for class 'epidataCS'
summary(object, ...)
## S3 method for class 'summary.epidataCS'
print(x, ...)

## S3 method for class 'epidataCS'
as.stepfun(x, ...)
```

Arguments

`events` a "SpatialPointsDataFrame" of cases with the following obligatory columns (in the `events@data` `data.frame`):

time time point of event. Will be converted to a numeric variable by `as.numeric`. There should be no concurrent events (but see `untie` for an ex post adjustment) and the event times must be covered by `stgrid`, i.e. belong to the time interval $(t_0, T]$, where t_0 is `min(stgrid$start)` and `T` is described below.

tile the spatial region (tile) where the event is located. This links to the tiles of `stgrid`.

type optional type of event in a marked `twinstim` model. Will be converted to a factor variable dropping unused levels. If missing, all events will be attribute the single type `"1"`.

eps.t maximum *temporal* influence radius (e.g. length of infectious period, time to culling, etc.); must be positive and may be `Inf`.

eps.s maximum *spatial* influence radius (e.g. 100 [km]); must be positive and may be `Inf`. A compact influence region mainly has computational advantages, but might also be plausible for specific applications.

The `data.frame` may contain columns with further marks of the events, e.g. sex, age of infected individuals, which may be used as epidemic covariates influencing infectiousness. Note that some auxiliary columns will be added at conversion whose names are reserved: `".obsInfLength"`, `".bdist"`, `".influenceRegion"`, and `".sources"`, as well as `"start"`, `"BLOCK"`, and all endemic covariates' names from `stgrid`.

`stgrid` a `data.frame` describing endemic covariates on a full spatio-temporal region x interval grid (e.g., district x week), which is a decomposition of the observation region `W` and period t_0, T. This means that for every combination of spatial region and time interval there must be exactly one row in this `data.frame`, that the union of the spatial tiles equals `W`, the union of the time intervals equals t_0, T,

and that regions (and intervals) are non-overlapping. There are the following obligatory columns:

tile ID of the spatial region (e.g., district ID). It will be converted to a factor variable (dropping unused levels if it already was one).

start, stop columns describing the consecutive temporal intervals (converted to numeric variables by `as.numeric`). The `start` time of an interval must be equal to the `stop` time of the previous interval. The `stop` column may be missing, in which case it will be auto-generated from the set of `start` values and `T`.

area area of the spatial region (`tile`). Be aware that the unit of this area (e.g., square km) must be consistent with the units of `W` and `events` (as specified in their `proj4string`s, if they have projected coordinates).

The remaining columns are endemic covariates. Note that the column name `"BLOCK"` is reserved (a column which will be added automatically for indexing the time intervals of `stgrid`).

`W` an object of class `"SpatialPolygons"` representing the observation region. It must have the same `proj4string` as `events` and all events must be within `W`. The function `simplify.owin` from package **spatstat** may be useful if polygonal operations take too long or memory is limited (see also the "Note" section below).

`qmatrix` a square indicator matrix (0/1 or `FALSE`/`TRUE`) for possible transmission between the event types. The matrix will be internally converted to `logical`. Defaults to an independent spread of the event types, i.e. the identity matrix.

`nCircle2Poly` accuracy (number of edges) of the polygonal approximation of a circle.

`T` end of observation period (i.e. last `stop` time of `stgrid`). Must be specified if the start but not the stop times are supplied in `stgrid` (=> auto-generation of `stop` times).

`clipper` polygon clipping engine to use for calculating the `.influenceRegions` of events (see the Value section below). Default is the **polyclip** package (called via `intersect.owin` from package **spatstat**). In **surveillance** <= 1.6-0, package **gpclib** was used, which has a restrictive license. This is no longer supported.

`verbose` logical indicating if status messages should be printed during input checking and `"epidataCS"` generation. The default is to do so in interactive R sessions.

`x` an object of class `"epidataCS"` or `"summary.epidataCS"`, respectively.

`n` a single integer. If positive, the first (`head`, `print`) / last (`tail`) `n` events are extracted. If negative, all but the `n` first/last events are extracted.

`digits` minimum number of significant digits to be printed in values.

`i,j,drop` arguments passed to the `[-method` for `SpatialPointDataFrames` for subsetting the `events` while retaining `stgrid` and `W`.
If `drop=TRUE` (the default), event types that completely disappear due to `i`-subsetting will be dropped, which reduces `qmatrix` and the factor levels of the `type` column.
By the `j` index, epidemic covariates can be removed from `events`.

`...` unused (arguments of the generics) with a few exceptions: The `print` method for `"epidataCS"` passes `...` to the `print.data.frame` method, and the `print` method for `"summary.epidataCS"` passes additional arguments to `print.table`.

`subset, select` arguments used to subset the `events` from an `"epidataCS"` object like in `subset.data.frame`.

`coords` logical indicating if the data frame of event marks returned by `marks(x)` should have the event coordinates appended as last columns. This defaults to `TRUE`.

`object` an object of class `"epidataCS"`.

Details

The function `as.epidataCS` is used to generate objects of class `"epidataCS"`, which is the data structure required for `twinstim` models.

The extraction method for class `"epidataCS"` ensures that the subsetted object will be valid, for instance, it updates the auxiliary list of potential transmission paths stored in the object. This `[`-method is also the basis for the `subset.epidataCS`-method, which is implemented similar to the `subset.data.frame`-method.

The `print` method for `"epidataCS"` prints some metadata of the epidemic, e.g., the observation period, the dimensions of the spatio-temporal grid, the types of events, and the total number of events. By default, it also prints the first `n = 6` rows of the `events`.

Value

An object of class `"epidataCS"` is a list containing the following components:

`events` a `"SpatialPointsDataFrame"` (see the description of the argument). The input `events` are checked for requirements and sorted chronologically. The columns are in the following order: obligatory event columns, event marks, the columns `BLOCK`, `start` and endemic covariates copied from `stgrid`, and finally, hidden auxiliary columns. The added auxiliary columns are:

- `.obsInfLength` observed length of the infectious period (being part [0,T]), i.e. `pmin(T-time, eps.t)`.
- `.sources` a list of numeric vectors of potential sources of infection (wrt the interaction ranges eps.s and eps.t) for each event. Row numbers are used as index.
- `.bdist` minimal distance of the event locations to the polygonal boundary `W`.
- `.influenceRegion` a list of influence regions represented by objects of the **spatstat** class `"owin"`. For each event, this is the intersection of `W` with a (polygonal) circle of radius `eps.s` centered at the event's location, shifted such that the event location becomes the origin. The list has `nCircle2Poly` set as an attribute.

`stgrid` a `data.frame` (see description of the argument). The spatio-temporal grid of endemic covariates is sorted by time interval (indexed by the added variable `BLOCK`) and region (`tile`). It is a full `BLOCK` x `tile` grid.

`W` a `"SpatialPolygons"` object representing the observation region.

`qmatrix` see the above description of the argument. The `storage.mode` of the indicator matrix is set to logical and the `dimnames` are set to the levels of the event types.

The nobs-method returns the number of events.

The head and tail methods subset the epidemic data using the extraction method ([), i.e. they return an object of class "epidataCS", which only contains (all but) the first/last n events.

For the "epidataCS" class, the method of the generic function marks defined by the **spatstat** package returns a data.frame of the event marks (actually also including time and location of the events), disregarding endemic covariates and the auxiliary columns from the events component of the "epidataCS" object.

The summary method (which has again a print method) returns a list of metadata, event data, the tables of tiles and types, a step function of the number of infectious individuals over time ($counter), i.e., the result of the as.stepfun-method for "epidataCS", and the number of potential sources of transmission for each event ($nSources) which is based on the given maximum interaction ranges eps.t and eps.s.

Note

The more detailed the observation region W is the longer will it take to fit a twinstim. It is often advisable to sacrifice some shape detail for speed by reducing polygon complexity using, e.g., the Douglas and Peucker (1973) reduction method available at MapShaper.org (Harrower and Bloch, 2006) or as function thinnedSpatialPoly in package **maptools**, or by passing via **spatstat**'s simplify.owin procedure.

Author(s)

Sebastian Meyer with documentation contributions by Michael Höhle and Mayeul Kauffmann.

References

Douglas, D. H. and Peucker, T. K. (1973): Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, **10**, 112-122.

Harrower, M. and Bloch, M. (2006): MapShaper.org: A Map Generalization Web Service. *IEEE Computer Graphics and Applications*, **26**(4), 22-27.
DOI-Link: http://dx.doi.org/10.1109/MCG.2006.85

Meyer, S., Elias, J. and Höhle, M. (2012): A space-time conditional intensity model for invasive meningococcal disease occurrence. *Biometrics*, **68**, 607-616.
DOI-Link: http://dx.doi.org/10.1111/j.1541-0420.2011.01684.x

Meyer, S. (2010): Spatio-Temporal Infectious Disease Epidemiology based on Point Processes. Master's Thesis, Ludwig-Maximilians-Universität München.
Available as http://epub.ub.uni-muenchen.de/11703/

See Also

plot.epidataCS for plotting, and animate.epidataCS for the animation of such an epidemic. There is also an update method for the "epidataCS" class. Models for "epidataCS" can be fitted with twinstim. It is also possible to convert the data to epidata objects (discrete space) for analysis with twinSIR (see as.epidata.epidataCS).

Examples

```
## load "imdepi" example data (which is an object of class "epidataCS")
data("imdepi")

## take a look at the data object
print(imdepi, n=5, digits=2)
s <- summary(imdepi)
s
plot(s$counter, xlab = "Time [days]",
     ylab="Number of infectious individuals",
     main=paste("Time course of the number of infectious individuals",
                "assuming an infectious period of 30 days", sep="\n"))
plot(table(s$nSources), xlab="Number of \"close\" infective individuals",
     ylab="Number of events",
     main=paste("Distribution of the number of potential sources",
                "assuming an interaction range of 200 km and 30 days",
                sep="\n"))
## the summary object contains further information
str(s)

## see the help page on the 'imdepi' dataset for more insight

## extraction methods subset the 'events' component
## (thereby taking care of the validity of the epidataCS object,
## for instance the hidden auxiliary column .sources and qmatrix)
imdepi[101:200, -match("sex", names(imdepi$events))]
tail(imdepi, n=4)           # reduce the epidemic to the last 4 events
subset(imdepi, type=="B")   # only consider event type B


### now reconstruct the object from its components

## events
events <- marks(imdepi)
coordinates(events) <- c("x", "y")  # promote to a "SpatialPointsDataFrame"
proj4string(events) <- proj4string(imdepi$events)       # ETRS89 projection
summary(events)

## endemic covariates
head(stgrid <- imdepi$stgrid[,-1])

## (Simplified) observation region (as SpatialPolygons)
load(system.file("shapes", "districtsD.RData", package="surveillance"),
     verbose = TRUE)

## plot observation region with events
plot(stateD, axes=TRUE); title(xlab="x [km]", ylab="y [km]")
points(events, pch=unclass(events$type), cex=0.5, col=unclass(events$type))
legend("topright", legend=levels(events$type), title="Type", pch=1:2, col=1:2)
## Not run:
    # similar to using the convenient plot-method for "epidataCS"
    plot(imdepi, "space")
```

```
## End(Not run)

## reconstruct the "imdepi" object from its components
myimdepi <- as.epidataCS(events = events, stgrid = stgrid,
                         W = stateD, qmatrix = diag(2), nCircle2Poly = 16)


## internal structure of an "epidataCS"-object
str(imdepi, max.level=4)
```

---

epidataCS_aggregate *Conversion (aggregation) of* "epidataCS" *to* "epidata" *or* "sts"

Description

Continuous-time continuous-space epidemic data stored in an object of class "epidataCS" can be aggregated in space or in space and time yielding an object of class "epidata" or "sts" for use of twinSIR or hhh4 modelling, respectively.

Usage

```
## aggregation in space and time over 'stgrid' for use of 'hhh4' models
epidataCS2sts(object, freq, start, neighbourhood,
              tiles = NULL, popcol.stgrid = NULL, popdensity = TRUE)

## aggregation in space for use of 'twinSIR' models
## S3 method for class 'epidataCS'
as.epidata(data, tileCentroids, eps = 0.001, ...)
```

Arguments

object, data	an object of class "epidataCS".
freq,start	see the description of the "sts" class.
neighbourhood	binary adjacency or neighbourhood-order matrix of the regions (tiles). If missing but tiles is given, a binary adjacency matrix will be auto-generated from tiles using functionality of the **spdep** package (see poly2adjmat). Since the "neighbourhood" slot in "sts" is actually optional, neighbourhood=NULL also works.
tiles	object inheriting from "SpatialPolygons" representing the regions in object$stgrid (column "tile"). It will become the "map" slot of the resulting "sts" object. Its row.names must match levels(object$stgrid$tile). If neighbourhood is provided, tiles is optional (not required for hhh4, but for plots of the resulting "sts" object).
popcol.stgrid	single character or numeric value indexing the column in object$stgrid which contains the population data (counts or densities, depending on the popdensity argument). This will become the "populationFrac" slot (optional).

popdensity	logical indicating if the column referenced by popcol.stgrid contains population densities or absolute counts.
tileCentroids	a coordinate matrix of the region centroids (i.e., the result of coordinates(tiles)). Its row names must match levels(data$stgrid$tile). This will be the coordinates used for the "population" (i.e., the tiles from "epidataCS") in the discrete-space twinSIR modelling.
eps	numeric scalar for breaking tied removal and infection times between different individuals (tiles), which might occur during conversion from "epidataCS" to "epidata". Rather dumb, this is simply done by subtracting eps from each tied removal time. One should consider other ways of breaking the tied event times.
...	unused (argument of the generic).

Details

Some comments on the conversion from "epidataCS" to "epidata": the conversion results into SIS epidemics only, i.e. the at-risk indicator is set to 1 immediately after recovery. A tile is considered infective if at least one individual within the tile is infective, otherwise it is susceptible. The lengths of the infectious periods are taken from data$events$eps.t. There will be no f columns in the resulting "epidata". These must be generated by a subsequent call to as.epidata with desired f.

Value

epidataCS2sts: an object of class "sts" representing the multivariate time-series of the number of cases aggregated over stgrid.

as.epidata.epidataCS: an object of class "epidata" representing an SIS epidemic in form of a multivariate point process (one for each region/tile).

Author(s)

Sebastian Meyer

See Also

epidata and twinSIR

linkS4class{sts} and hhh4.

Examples

```
data("imdepi")
load(system.file("shapes", "districtsD.RData", package="surveillance"))

## convert imdepi point pattern into multivariate time series
imdepi_sts <- epidataCS2sts(imdepi, freq=12, start=c(2002,1),
                            neighbourhood=NULL, # not needed here
                            tiles=districtsD)


## compare plots of monthly number of cases
```

```
opar <- par(mfrow=c(2,1))
suppressWarnings(plot(imdepi, "time", breaks=c(0,unique(imdepi$stgrid$stop))))
plot(imdepi_sts, type=observed~time, legend.opts=NULL)
par(opar)

## plot number of cases by district
plot(imdepi_sts, type=observed~1|unit, labels=FALSE)


## also test conversion to an SIS event history ("epidata") of the "tiles"
if (requireNamespace("intervals")) {
    imdepi_short <- subset(imdepi, time < 50)
    imdepi_short$stgrid <- subset(imdepi_short$stgrid, start < 50)
    imdepi_epidata <- as.epidata(imdepi_short,
                                 tileCentroids=coordinates(districtsD))
    summary(imdepi_epidata)
}
```

`epidataCS_animate` *Spatio-Temporal Animation of a Continuous-Time Continuous-Space Epidemic*

Description

Function for the animation of continuous-time continuous-space epidemic data, i.e. objects inheriting from class `"epidataCS"`. There are three types of animation, see argument `time.spacing`. Besides the on-screen plotting in the interactive R session, it is possible and recommended to redirect the animation to an off-screen graphics device using the contributed R package **animation**. For instance, the animation can be watched and navigated in a web browser via `saveHTML` (see Examples).

Usage

```
## S3 method for class 'epidataCS'
animate(object, interval = c(0,Inf), time.spacing = NULL,
        nmax = NULL, sleep = NULL, legend.opts = list(), timer.opts = list(),
        pch = 15:18, col.current = "red", col.I = "#C16E41",
        col.R = "#B3B3B3", col.influence = NULL,
        main = NULL, verbose = interactive(), ...)
```

Arguments

`object`	an object inheriting from class `"epidataCS"`.
`interval`	time range of the animation.
`time.spacing`	time interval for the animation steps. If `NULL` (the default), the events are plotted sequentially by producing a snapshot at every time point where an event occured. Thus, it is just the *ordering* of the events, which is shown.

To plot the appearance of events proportionally to the exact time line, `time.spacing` can be set to a numeric value indicating the period of time between consecutive snapshots. Then, for each time point in `seq(0, end, by = time.spacing)` the current state of the epidemic can be seen and an additional timer indicates the current time (see `timer.opts` below).
If `time.spacing = NA`, then the time spacing is automatically determined in such a way that `nmax` snapshots result. In this case, `nmax` must be given a finite value.

`nmax` — maximum number of snapshots to generate. The default `NULL` means to take the value from `ani.options("nmax")` if the **animation** package is available, and no limitation (`Inf`) otherwise.

`sleep` — numeric scalar specyfing the artificial pause in seconds between two time points (using `Sys.sleep`), or `NULL` (default), when this is taken from `ani.options("interval")` if the **animation** package is available, and set to 0.1 otherwise. Note that `sleep` is ignored on non-interactive devices (see `dev.interactive`), e.g., if generating an animation inside **animation**'s `saveHTML`.

`pch, col` — vectors of length equal to the number of event types specifying the point symbols and colors for events to plot (in this order). The vectors are recycled if necessary.

`legend.opts` — either a list of arguments passed to the `legend` function or `NULL` (or `NA`), in which case no legend will be plotted. All necessary arguments have sensible defaults and need not be specified.

`timer.opts` — either a list of arguments passed to the `legend` function or `NULL` (or `NA`), in which case no timer will be plotted. All necessary arguments have sensible defaults and need not be specified, i.e.

```
x: "bottomright"
title: "time"
box.lty: 0
adj: c(0.5,0.5)
inset: 0.01
bg: "white"
```

Note that the argument `legend`, which is the current time of the animation, can not be modified.

`col.current` — color of events when occuring (new).

`col.I` — color once infectious.

`col.R` — color event has once "recovered". If `NA`, then recovered events will not be shown.

`col.influence` — color with which the influence region is drawn. Use `NULL` (default) if no influence regions should be drawn.

`main` — optional main title placed above the map.

`verbose` — logical specifying if a (textual) progress bar should be shown during snapshot generation. This is especially useful if the animation is produced within `saveHTML` or similar.

`...` — further graphical parameters passed to the plot-method of the `SpatialPolygons-class`.

Author(s)

Sebastian Meyer with documentation contributions by Michael Höhle

See Also

`plot.epidataCS` for plotting the numbers of events by time (aggregated over space) or the locations of the events in the observation region `W` (aggregated over time).

The contributed R package **animation**.

Examples

```
data("imdepi")
imdepiB <- subset(imdepi, type == "B")

# Animate the first year of type B with a step size of 7 days
animate(imdepiB, interval=c(0,365), time.spacing=7, nmax=Inf, sleep=0.1)

# Sequential animation of type B events during the first year
animate(imdepiB, interval=c(0,365), time.spacing=NULL, sleep=0.1)

# Animate the whole time range but with nmax=20 snapshots only
animate(imdepiB, time.spacing=NA, nmax=20, sleep=0.1)


# Such an animation can be saved in various ways using the tools of
# the animation package, e.g., saveHTML()
if (require("animation")) {
  oldwd <- setwd(tempdir())  # to not clutter up the current working dir
  saveHTML(animate(imdepiB, interval = c(0,365), time.spacing = 7),
           nmax = Inf, interval = 0.2, loop = FALSE,
           title = "Animation of the first year of type B events")
  setwd(oldwd)
}
```

---

epidataCS_plot *Plotting the Events of an Epidemic over Time and Space*

Description

The `plot` method for class `"epidataCS"` either plots the number of events along the time axis (`epidataCSplot_time`) as a `hist()`, or the locations of the events in the observation region `W` (`epidataCSplot_space`).

Usage

```
## S3 method for class 'epidataCS'
plot(x, aggregate = c("time", "space"), subset, ...)

epidataCSplot_time(x, subset, t0.Date = NULL, freq = TRUE,
                   col = rainbow(nTypes), cumulative = list(),
                   add = FALSE, mar = NULL, xlim = NULL, ylim = NULL,
                   xlab = "Time", ylab = NULL, main = NULL,
                   panel.first = abline(h=axTicks(2), lty=2, col="grey"),
                   legend.types = list(), ...)

epidataCSplot_space(x, subset,
                    cex.fun = sqrt, points.args = list(), add = FALSE,
                    legend.types = list(), legend.counts = list(), ...)
```

Arguments

x	an object of class "epidataCS".
aggregate	character, one of "time" and "space", referring to the specific plot functions epidataCSplot_time and epidataCSplot_time, respectively. For "time", the number of events over time is plotted as hist (or hist.Date). For "space", the observation region x$W and the locations of the events therein are plotted.
subset	logical expression indicating a subset of events to consider for plotting: missing values are taken as false. Note that the expression is evaluated in the data frame of event marks (marks(x)), which means that column names can be referred to by name (like in subset.data.frame).
...	in the basic plot-method further arguments are passed to the aggregate-specific plot function. In epidataCSplot_time, further graphical parameters are passed to hist or hist.Date, respectively. In epidataCSplot_space, further arguments are passed to the plot-method for "SpatialPolygons", which draws the observation region x$W.
t0.Date	the beginning of the observation period t0 = x$stgrid$start[1] as a "Date" (or anything coercible by as.Date without further arguments), enabling a nice x-axis using hist.Date and sensible breaks of the histogram (which must also be specified in this case, e.g., breaks="months"). The event times then equal t0.Date + as.integer(x$events$time - t0), i.e. possible fractional parts of the event times are removed (which ensures that using breaks = "months" or other automatic types always works).
freq	see hist, defaults to TRUE.
col	fill colour for the bars of the histogram, defaults to the vector of rainbow colours.
cumulative	if a list (of style options), lines for the cumulative number of events (per type) will be added to the plot. Possible options are axis (logical), lab (axis label), maxat (single integer affecting the axis range), lwd, col, and offset (a numeric vector of length the number of types).
add	logical (default: FALSE) indicating if the plot should be added to an existing window.

mar	see par. The default (NULL) is mar <- par("mar"), with mar[4] <- mar[2] if an axis is requested for the cumulative numbers.
xlim,ylim	NULL provides automatic axis limits.
xlab,ylab	axis labels (with sensible defaults).
main	main title of the plot (defaults to no title).
panel.first	expression that should be evaluated after the plotting window has been set up but before the histogram is plotted. Defaults to adding horizontal grid lines.
legend.types	if a list (of arguments for legend), a legend for the event types is added to the plot.
cex.fun	function which takes a vector of counts of events at each unique location and returns a (vector of) cex value(s) for the sizes of the corresponding points. Defaults to the sqrt() function, which for the default circular pch=1 means that the area of each point is proportional to the number of events at its location.
points.args	a list of (type-specific) graphical parameters for points, specifically pch, lwd, and col, which are all recycled to give the length nlevels(x$events$type). In contrast, a possible cex element should be scalar (default: 0.5) and multiplies the sizes obtained from cex.fun.
legend.counts	if a list (of arguments for legend), a legend illustrating the effect of cex.fun is added to the plot. This list may contain a special element counts, which is an integer vector specifying the counts to illustrate.

Value

For aggregate="time" (i.e., epidataCSplot_time) the data of the histogram (as returned by hist), and for aggregate="space" (i.e., epidataCSplot_space) NULL, invisibly.

Author(s)

Sebastian Meyer

See Also

animate.epidataCS

Examples

```
data("imdepi")

## show the occurrence of events along the time axis (-> histogram)
plot(imdepi, "time", main = "Histogram of event time points")

## show the distribution in space
plot(imdepi, "space", lwd=2)
```

epidataCS_update *Update method for* "epidataCS"

Description

The update method for the "epidataCS" class may be used to modify the hyperparameters ϵ (eps.t) and δ (eps.s), the indicator matrix qmatrix of possible ways of transmission between the event types, and the numerical accuracy nCircle2Poly of the polygonal representation of a circle. The update method will also update the auxiliary information contained in an "epidataCS" object accordingly, e.g., the vector of potential sources of each event, or the polygonal representation of the influence region.

Usage

```
## S3 method for class 'epidataCS'
update(object, eps.t, eps.s, qmatrix, nCircle2Poly, ...)
```

Arguments

object	an object of class "epidataCS".
eps.t	numeric vector of length the number of events in object$events. The event data column eps.t specifies the maximum temporal influence radius (e.g., length of infectious period, time to culling, etc.) of the events.
eps.s	numeric vector of length the number of events in object$events. The event data column eps.s specifies the maximum spatial influence radius of the events.
qmatrix	square indicator matrix (0/1 or TRUE/FALSE) for possible transmission between the event types.
nCircle2Poly	accuracy (number of edges) of the polygonal approximation of a circle.
...	unused (argument of the generic).

Value

The updated "epidataCS" object.

Author(s)

Sebastian Meyer

See Also

class "epidataCS".

Examples

```
data("imdepi")

## assume different interaction ranges and simplify polygons
imdepi2 <- update(imdepi, eps.t = 20, eps.s = Inf, nCircle2Poly = 16)

(s <- summary(imdepi))
(s2 <- summary(imdepi2))
## The update reduced the number of infectives (along time)
## because the length of the infectious periods is reduced. It also
## changed the set of potential sources of transmission for each
## event, since the interaction is shorter in time but wider in space
## (eps.s=Inf means interaction over the whole observation region).
```

---

epidata_animate *Spatio-Temporal Animation of an Epidemic*

Description

Function for the animation of epidemic data, i.e. objects inheriting from class `"epidata"`. This only works with 1- or 2-dimensional coordinates and is not useful if some individuals share the same coordinates (overlapping). There are two types of animation, see argument `time.spacing`. Besides the direct plotting in the R session, it is also possible to generate a sequence of graphics files to create animations outside R.

Usage

```
## S3 method for class 'summary.epidata'
animate(object, main = "An animation of the epidemic",
        pch = 19, col = c(3, 2, gray(0.6)), time.spacing = NULL,
        sleep = quote(5/.nTimes), legend.opts = list(), timer.opts = list(),
        end = NULL, generate.snapshots = NULL, ...)

## S3 method for class 'epidata'
animate(object, ...)
```

Arguments

`object`	an object inheriting from class `"epidata"` or `"summary.epidata"`. In the former case, its summary is calculated and the function continues as in the latter case, passing all `...` arguments to the `summary.epidata` method.
`main`	a main title for the plot, see also `title`.
`pch, col`	vectors of length 3 specifying the point symbols and colors for susceptible, infectious and removed individuals (in this order). The vectors are recycled if necessary. By default, susceptible individuals are marked as filled green circles, infectious individuals as filled red circles and removed individuals as filled gray

circles. Note that the symbols are iteratively drawn (overlayed) in the same plotting region as time procedes. For information about the possible values of pch and col, see the help pages of points and par, respectively.

time.spacing — time interval for the animation steps. If NULL (the default), the events are plotted one by one with pauses of sleep seconds. Thus, it is just the *ordering* of the events, which is shown. To plot the appearance of events proportionally to the exact time line, time.spacing can be set to a numeric value indicating the period of time between consecutive plots. Then, for each time point in seq(0, end, by = time.spacing) the current state of the epidemic can be seen and an additional timer indicates the current time (see timer.opts below). The argument sleep will be the artificial pause in seconds between two of those time points.

sleep — time in seconds to Sys.sleep before the next plotting event. By default, each artificial pause is of length 5/.nTimes seconds, where .nTimes is the number of events (infections and removals) of the epidemic, which is evaluated in the function body. Thus, for time.spacing = NULL the animation has a duration of approximately 5 seconds. In the other case, sleep is the duration of the artificial pause between two time points. Note that sleep is ignored on non-interactive devices (see dev.interactive)

legend.opts — either a list of arguments passed to the legend function or NULL (or NA), in which case no legend will be plotted. All necessary arguments have sensible defaults and need not be specified, i.e.

x: "topright"

legend: c("susceptible", "infectious", "removed")

pch: same as argument pch of the main function

col: same as argument col of the main function

timer.opts — either a list of arguments passed to the legend function or NULL (or NA), in which case no timer will be plotted. All necessary arguments have sensible defaults and need not be specified, i.e.

x: "bottomright"

title: "time"

box.lty: 0

adj: c(0.5,0.5)

inset: 0.01

bg: "white"

Note that the argument legend, which is the current time of the animation, can not be modified.

end — ending time of the animation in case of time.spacing not being NULL. By default (NULL), time stops after the last event.

generate.snapshots — By default (NULL), the animation is not saved to image files but only shown on the on-screen device. In order to do that, time.spacing must not be NULL, and there are two options:
If the framework of the **animation** package should be used, i.e. the animate-call is passed as the expr argument to one of the save* functions of the **animation**

package, then set `generate.snapshots = img.name`, where `img.name` is the base name for the generated images (the same as passed to the `save*` function). The path and format (type, width, height) for the generated images is derived from `ani.options`. See the last example below.
Alternatively, `generate.snapshots` may be a list of arguments passed to the function `dev.print`, which then is executed at each time point of the grid defined by `time.spacing`. Essentially, this is used for saving the produced snapshots to files, e.g.

```
generate.snapshots =    list(device=pdf, file=quote(paste("epidemic_",sprintf(form,t
```

will store the animation steps in pdf-files in the current working directory, where the file names each end with the time point represented by the corresponding plot. Because the variables `tp` and `form` should only be evaluated inside the function the `file` argument is quoted. Alternatively, the file name could also make use of the internal plot index `i`, e.g., use `file=quote(paste("epidemic",i,".pdf",sep=""))`.

`...` further graphical parameters passed to the basic call of `plot`, e.g. `las`, `cex.axis` (etc.) and `mgp`.

Author(s)

Sebastian Meyer

See Also

`summary.epidata` for the data, on which the plot is based. `plot.epidata` for plotting the evolution of an epidemic by the numbers of susceptible, infectious and removed individuals.

The contributed R package **animation**.

Examples

```
data("fooepidata")
(s <- summary(fooepidata))

# plot the ordering of the events only
animate(s, sleep=0.01)   # or: animate(fooepidata)

# with timer (animate only up to t=10)
animate(s, time.spacing=0.1, end=10, sleep=0.01)


# Such an animation can be saved in various ways using tools of
# the animation package, e.g., saveHTML()
if (require("animation")) {
  oldwd <- setwd(tempdir())  # to not clutter up the current working dir
  saveHTML({
    par(bg="white")  # default "transparent" is grey in some browsers
    animate(s, time.spacing=1, sleep=0, generate.snapshots="epiani")
  }, use.dev=FALSE, img.name="epiani", ani.width=600, interval=0.5)
  setwd(oldwd)
}
```

epidata_intersperse *Impute Blocks for Extra Stops in* "epidata" *Objects*

Description

This function modifies an object inheriting from class "epidata" such that it features the specified stop time points. For this purpose, the time interval in the event history into which the new stop falls will be splitted up into two parts, one block for the time period until the new stop – where no infection or removal occurs – and the other block for the time period from the new stop to the end of the original interval.
Main application is to enable the use of knots in twinSIR, which are not existing stop time points in the "epidata" object.

Usage

```
intersperse(epidata, stoptimes, verbose = FALSE)
```

Arguments

epidata	an object inheriting from class "epidata".
stoptimes	a numeric vector of time points inside the observation period of the epidata.
verbose	logical indicating if a txtProgressBar should be shown while inserting blocks for extra stoptimes.

Value

an object of the same class as epidata with additional time blocks for any new stoptimes.

Author(s)

Sebastian Meyer

See Also

as.epidata.epidataCS where this function is used.

Examples

```
data("fooepidata")
subset(fooepidata, start < 25 & stop > 25, select = 1:7)
nrow(fooepidata)
moreStopsEpi <- intersperse(fooepidata, c(25,75))
nrow(moreStopsEpi)
subset(moreStopsEpi, stop == 25 | start == 25, select = 1:7)
```

epidata_plot *Plotting the Evolution of an Epidemic*

Description

Functions for plotting the evolution of epidemics. The plot methods for classes "epidata" and "summary.epidata" plots the numbers of susceptible, infectious and recovered (= removed) individuals by step functions along the time axis. The function stateplot shows individual state changes along the time axis.

Usage

```
## S3 method for class 'summary.epidata'
plot(x,
     lty = c(2, 1, 3), lwd = 2,
     col = c("#1B9E77", "#D95F02", "#7570B3"), col.hor = col, col.vert = col,
     xlab = "Time", ylab = "Number of individuals",
     xlim = NULL, ylim = NULL, legend.opts = list(), do.axis4 = NULL,
     panel.first = grid(), rug.opts = list(),
     which.rug = c("infections", "removals", "susceptibility", "all"), ...)
## S3 method for class 'epidata'
plot(x, ...)

stateplot(x, id, ...)
```

Arguments

x	an object inheriting from class "epidata" or "summary.epidata". In the former case, its summary is calculated and the function continues as in the latter case. The plot method for class "epidata" is a simple wrapper for plot.summary.epidata implemented as plot(summary(x, ...)).
lty, lwd	vectors of length 3 containing the line types and widths, respectively, for the numbers of susceptible, infectious and removed individuals (in this order). By default, all lines have width 1 and the line types are dashed (susceptible), solid (infectious) and dotted (removed), respectively. To omit the drawing of a specific line, just set the corresponding entry in lty to 0. The vectors are recycled if necessary. For information about the different lty and lwd codes, see the help pages of par.
col, col.hor, col.vert	vectors of length 3 containing the line colors for the numbers of susceptible, infectious and removed individuals (in this order). col.hor defines the color for the horizontal parts of the step function, whilst col.vert defines the color for its vertical parts. The argument col is just short for col.hor = col and col.vert = col. The default col vector corresponds to brewer.pal("Dark2",n=3) from the **RColorBrewer** package. The vectors are recycled if necessary. For information about the possible values of col, see the help pages of par.

`xlab, ylab` axis labels, default to "Time" and "Number of individuals", respectively.

`xlim, ylim` the x and y limits of the plot in the form `c(xmin, xmax)` and `c(ymin, ymax)`, respectively. By default, these are chosen adequately to fit the time range of the epidemic and the number of individuals.

`legend.opts` if this is a list (of arguments for the `legend` function), a legend will be plotted. The defaults are as follows:

`x`: `"topright"`

`inset`: `c(0,0.02)`

`legend`: `c("susceptible", "infectious", "removed")`

`lty`,`lwd`,`col`: same as the arguments `lty`, `lwd`, and `col.hor` of the main function

`bty`: `"n"`

`do.axis4` logical indicating if the final numbers of susceptible and removed individuals should be indicated on the right axis. The default `NULL` means `TRUE`, if `x` represents a SIR epidemic and `FALSE` otherwise, i.e. if the epidemic is SI, SIS or SIRS.

`panel.first` an expression to be evaluated after the plot axes are set up but before any plotting takes place. By default, a standard grid is drawn.

`rug.opts` either a list of arguments passed to the function `rug` or `NULL` (or `NA`), in which case no `rug` will be plotted. By default, the argument `ticksize` is set to 0.02, `col` is set to the color according to `which.rug` (black if this is `"all"`), and `quiet` is set to `TRUE`. Note that the argument `x`, which contains the locations for the `rug` is fixed internally and can not be modified. The argument `which.rug` (see below) determines the locations to mark.

`which.rug` By default, tick marks are drawn at the time points of infections. Alternatively, one can choose to mark only `"removals"`, `"susceptibilities"` (i.e. state change from R to S) or `"all"` events.

`id` single character string or factor of length 1 specifying the individual for which the `stateplot` should be established.

`...` For `plot.summary.epidata`: further graphical parameters passed to `plot`, `lines` and `axis`, e.g. `main`, `las`, `cex.axis` (etc.) and `mgp`.
For `plot.epidata`: arguments passed to `plot.summary.epidata`.
For `stateplot`: arguments passed to `plot.stepfun` or `plot.function` (if `id` had no events during the observation period). By default, `xlab="time"`, `ylab="state"`, `xlim=attr(x,"timeRange")`, `xaxs="i"` and `do.points=FALSE`.

Value

`plot.summary.epidata` (and `plot.epidata`) invisibly returns the matrix used for plotting, which contains the evolution of the three counters.
`stateplot` invisibly returns the function, which was plotted, typically of class `"stepfun"`, but maybe of class `"function"`, if no events have been observed for the individual in question (then the function always returns the initial state). The vertical axis of `stateplot` can range from 1 to 3, where 1 corresponds to *S*uscepible, 2 to *I*nfectious and 3 to *R*emoved.

Author(s)

Sebastian Meyer

See Also

summary.epidata for the data, on which the plots are based. animate.epidata for the animation of epidemics.

Examples

```
data("fooepidata")
s <- summary(fooepidata)

# evolution of the epidemic
par(las = 1)
plot(s)

# stateplot
stateplot(s, id = "15", main = "Some individual event paths")
stateplot(s, id = "1", add = TRUE, col = 2)
stateplot(s, id = "20", add = TRUE, col = 3)
legend("topright", legend = c(15, 1, 20), title = "id", lty = 1, col = 1:3,
       inset = 0.1)
```

---

epidata_summary *Summarizing an Epidemic*

Description

The summary method for class "epidata" gives an overview of the epidemic. Its print method shows the type of the epidemic, the time range, the total number of individuals, the initially and never infected individuals and the size of the epidemic. An excerpt of the returned counters data frame is also printed (see the Value section below).

Usage

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

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

Arguments

object	an object inheriting from class "epidata".
x	an object inheriting from class "summary.epidata", i.e. an object returned by the function summary.epidata.
...	unused (argument of the generic).

Value

A list with the following components:

`type`	character string. Compartmental type of the epidemic, i.e. one of "SIR", "SI", "SIS" or "SIRS".
`size`	integer. Size of the epidemic, i.e. the number of initially susceptible individuals, which became infected during the course of the epidemic.
`initiallyInfected`	factor (with the same levels as the `id` column in the `"epidata"` object). Set of initially infected individuals.
`neverInfected`	factor (with the same levels as the `id` column in the `"epidata"` object). Set of never infected individuals, i.e. individuals, which were neither initially infected nor infected during the course of the epidemic.
`coordinates`	numeric matrix of individual coordinates with as many rows as there are individuals and one column for each spatial dimension. The row names of the matrix are the `id`s of the individuals.
`byID`	data frame with time points of infection and optionally removal and re-susceptibility (depending on the `type` of the epidemic) ordered by `id`. If an event was not observed, the corresponding entry is missing.
`counters`	data frame containing all events (S, I and R) ordered by time. The columns are `time`, `type` (of event), corresponding `id` and the three counters `nSusceptible`, `nInfectious` and `nRemoved`. The first row additionally shows the counters at the beginning of the epidemic, where the `type` and `id` column contain missing values.

Author(s)

Sebastian Meyer

See Also

`as.epidata` for generating objects of class `"epidata"`.

Examples

```
data("fooepidata")
s <- summary(fooepidata)
s          # uses the print method for summary.epidata
names(s)   # components of the list 's'

# positions of the individuals
plot(s$coordinates)

# events by id
head(s$byID)
```

`estimateGLRNbHook` *Hook function for in-control mean estimation*

Description

Allows the user to specify his own estimation routine for the in-control mean of `algo.glrpois` Needs to work for GLRNbHook

Usage

```
estimateGLRNbHook()
```

Details

This hook function allows the user to customize the behaviour of the algorithm.

Value

A list

`mod`	resulting model of a call of `glm.nb`
`range`	vector of length as `range` containing the predicted values

Author(s)

M. Hoehle

See Also

`algo.glrnb` and `algo.glrpois`

Examples

```
## Not run:
estimateGLRNbHook <- function() {
  #Fetch control object from parent
  control <- parent.frame()$control
  #The period
  p <- parent.frame()$disProgObj$freq
  #Current range to perform surveillance on
  range <- parent.frame()$range

  #Define training & test data set (the rest)
  train <- 1:(range[1]-1)
  test <- range

  #Perform an estimation based on all observations before timePoint
  #Event better - don't do this at all in the algorithm - force
  #user to do it himself - coz its a model selection problem
```

```
  data <- data.frame(y=parent.frame()$disProgObj$observed[t],t=train)
  #Build the model equation
  formula <- "y ~ 1 "
  if (control$mu0Model$trend) { formula <- paste(formula," + t",sep="") }
  for (s in 1:control$mu0Model$S) {
    formula <- paste(formula,"+cos(2*",s,"*pi/p*t)+ sin(2*",s,"*pi/p*t)",sep="")
  }
  #Fit the GLM
  m <- eval(substitute(glm.nb(form,data=data),
                       list(form=as.formula(formula))))

  #Predict mu_{0,t}
  return(as.numeric(predict(m,newdata=data.frame(t=range),type="response")))
}

## End(Not run)
```

`estimateGLRPoisHook` *Hook function for in-control mean estimation*

Description

Allows the user to specify his own estimation routine for the in-control mean of `algo.glrpois`

Usage

```
estimateGLRPoisHook()
```

Details

This hook function allows the user to customize the behaviour of the algorithm.

Value

A vector of length as `range` containing the predicted values.

Author(s)

M. Hoehle

See Also

`algo.glrpois`

Examples

```
## Not run:
estimateGLRPoisHook <- function() {
  #Fetch control object from parent
  control <- parent.frame()$control
  #The period
  p <- parent.frame()$disProgObj$freq
  #Current range to perform surveillance on
  range <- parent.frame()$range

  #Define training & test data set (the rest)
  train <- 1:(range[1]-1)
  test <- range

  #Perform an estimation based on all observations before timePoint
  #Event better - don't do this at all in the algorithm - force
  #user to do it himself - coz its a model selection problem
  data <- data.frame(y=parent.frame()$disProgObj$observed[t],t=train)
  #Build the model equation
  formula <- "y ~ 1 "
  if (control$mu0Model$trend) { formula <- paste(formula," + t",sep="") }
  for (s in 1:control$mu0Model$S) {
    formula <- paste(formula,"+cos(2*",s,"*pi/p*t)+ sin(2*",s,"*pi/p*t)",sep="")
  }
  #Fit the GLM
  m <- eval(substitute(glm(form,family=poisson(),data=data),list(form=as.formula(formula))))

  #Predict mu_{0,t}
  return(as.numeric(predict(m,newdata=data.frame(t=range),type="response")))
}

## End(Not run)
```

`farringtonFlexible` *Surveillance for an univariate count data time series using the improved Farrington method described in Noufaily et al. (2012).*

Description

The function takes `range` values of the surveillance time series `sts` and for each time point uses a Poisson GLM with overdispersion to predict an upper bound on the number of counts according to the procedure by Farrington et al. (1996) and by Noufaily et al. (2012). This bound is then compared to the observed number of counts. If the observation is above the bound, then an alarm is raised.

Usage

```
farringtonFlexible(sts, control = list(range = NULL, b = 3, w = 3,
                                       reweight = TRUE,
```

```
                                    weightsThreshold = 2.58,
                                    verbose = FALSE,glmWarnings = TRUE,
                                    alpha = 0.01, trend = TRUE,
                                    pThresholdTrend = 0.05, limit54=c(5,4),
                                    powertrans="2/3",
                                    fitFun="algo.farrington.fitGLM.flexible",
                                    populationOffset = FALSE,
                                    noPeriods = 1, pastWeeksNotIncluded = 26,
                                    thresholdMethod = "delta"))
```

Arguments

`sts` object of class sts (including the `observed` and the `state` time series)

`control` Control object given as a `list` containing the following components:

`range` Specifies the index of all timepoints which should be tested. If range is `NULL` all possible timepoints are used.

`b` How many years back in time to include when forming the base counts.

`w` Window's half-size, i.e. number of weeks to include before and after the current week in each year.

`reweight` Boolean specifying whether to perform reweighting step.

`weightsThreshold` Defines the threshold for reweighting past outbreaks using the Anscombe residuals (1 in the original method, 2.58 advised in the improved method).

`verbose` Boolean specifying whether to show extra debugging information.

`glmWarnings` Boolean specifying whether to print warnings from the call to `glm`.

`alpha` An approximate (one-sided) $(1-\alpha)\cdot 100\%$ prediction interval is calculated unlike the original method where it was a two-sided interval. The upper limit of this interval i.e. the $(1-\alpha)\cdot 100\%$ quantile serves as an upperbound.

`trend` Boolean indicating whether a trend should be included and kept in case the conditions in the Farrington et. al. paper are met (see the results). If `false` then NO trend is fit.

`pThresholdTrend` Threshold for deciding whether to keep trend in the model (0.05 in the original method, 1 advised in the improved method).

`limit54` Vector containing two numbers: `cases` and `period`. To avoid alarms in cases where the time series only has about almost no cases in the specific week the algorithm uses the following heuristic criterion (see Section 3.8 of the Farrington paper) to protect against low counts: no alarm is sounded if fewer than `cases` $= 5$ reports were received in the past `period` $= 4$ weeks. `limit54=c(cases,period)` is a vector allowing the user to change these numbers. Note: As of version 0.9-7 of the package the term "last" period of weeks includes the current week - otherwise no alarm is sounded for horrible large numbers if the four weeks before that are too low.

`powertrans` Power transformation to apply to the data if the threshold is to be computed with the method described in Farrington et al. (1996. Use

either "2/3" for skewness correction (Default), "1/2" for variance stabilizing transformation or "none" for no transformation.

`fitFun` String containing the name of the fit function to be used for fitting the GLM. The only current option is "algo.farrington.fitGLM.flexible".

`populationOffset` Boolean specifying whether to include a population offset in the GLM. The slot `sts@population` gives the population vector.

`noPeriods` Number of levels in the factor allowing to use more baseline. If equal to 1 no factor variable is created, the set of reference values is defined as in Farrington et al (1996).

`pastWeeksNotIncluded` Number of past weeks to ignore in the calculation.

`thresholdMethod` Number of past weeks to ignore in the calculation. Options are "delta" for the method described in Farrington et al. (1996), "Noufaily" for the method described in Noufaily et al. (2012), and "muan" for the method extended from Noufaily et al. (2012).

Details

The following steps are perfomed according to the Farrington et al. (1996) paper.

1. Fit of the initial model with intercept, time trend if `trend` is `TRUE`, seasonal factor variable if `noPeriod` is bigger than 1, and population offset if `populationOffset` is `TRUE`. Initial estimation of mean and overdispersion.
2. Calculation of the weights omega (correction for past outbreaks) if `reweighting` is `TRUE`. The threshold for reweighting is defined in `control`.
3. Refitting of the model
4. Revised estimation of overdispersion
5. Omission of the trend, if it is not significant
6. Repetition of the whole procedure
7. Calculation of the threshold value using the model to compute a quantile of the predictive distribution. The method used depends on `thresholdMethod`, this can either be:

   "delta" One assumes that the prediction error (or a transformation of the prediction error, depending on `powertrans`), is normally distributed. The threshold is deduced from a quantile of this normal distribution using the variance and estimate of the expected count given by GLM, and the delta rule. The procedure takes into account both the estimation error (variance of the estimator of the expected count in the glm) and the prediction error (variance of the prediction error). This is the suggestion in Farrington et al. (1996).

   "nbPlugin" One assumes that the new count follows a negative binomial distribution parameterized by the expected count and the overdispersion estimated in the GLM. The threshold is deduced from a quantile of this discrete distribution. This process disregards the estimation error, though. This method was used in Noufaily, et al. (2012).

   "muan" One also uses the assumption of the negative binomial sampling distribution but does not plug in the estimate of the expected count from the GLM, instead one uses a quantile from the asymptotic normal distribution of the expected count estimated in the GLM; in order to take into account both the estimation error and the prediction error.
8. Computation of exceedance score

Warning: monthly data containing the last day of each month as date should be analysed with `epochAsDate=FALSE` in the `sts` object. Otherwise February makes it impossible to find some reference time points.

Value

An object of class `sts` with the slots `upperbound` and `alarm` filled by appropriate output of the algorithm. The slot `control` is the usual list but with more items all of length `length(range)`:

trend is a vector of Booleans indicating whether a time trend was fitted for this time point.

trendVector is a vector giving the coefficient of the time trend in the GLM for this time point. If no trend was fitted it is equal to NA.

pvalue is a vector giving the probability of observing a value at least equal to the observation under the null hypothesis .

expected is a vector giving the expectation of the predictive distribution for each timepoint. It is only reported if the conditions for raising an alarm are met, e.g enoughCases.

mu0Vector is a vector giving what is inputed in the negative binomial distribution to get the upperbound as a quantile (either a plug-in from the glm or a quantile from the asymptotic normal distribution of the estimator)

phiVector is a vector giving the overdispersion of the glm at each timepoint.

Author(s)

M. Salmon, M. Höhle

References

A statistical algorithm for the early detection of outbreaks of infectious disease, Farrington, C.P., Andrews, N.J, Beale A.D. and Catchpole, M.A. (1996), J. R. Statist. Soc. A, 159, 547-563.

An improved algorithm for outbreak detection in multiple surveillance systems, Noufaily, A., Enki, D.G., Farrington, C.P., Garthwaite, P., Andrews, N.J., Charlett, A. (2012), Statistics in Medicine, published online.

See Also

`algo.farrington.fitGLM`,`algo.farrington.threshold`

Examples

```
### DATA I/O ###
#Read Salmonella Agona data
data("salmonella.agona")

# Create the corresponding sts object from the old disProg object
salm <- disProg2sts(salmonella.agona)

### RUN THE ALGORITHMS WITH TWO DIFFERENT SETS OF OPTIONS ###
# Farrington with old options
```

```
control1 <-  list(range=(260:312),
 noPeriods=1,populationOffset=FALSE,
 fitFun="algo.farrington.fitGLM.flexible",
 b=4,w=3,weightsThreshold=1,
 pastWeeksNotIncluded=3,
 pThresholdTrend=0.05,trend=TRUE,
 thresholdMethod="delta",alpha=0.1)
control2<-   list(range=(260:312),
 noPeriods=10,populationOffset=FALSE,
 fitFun="algo.farrington.fitGLM.flexible",
 b=4,w=3,weightsThreshold=2.58,
 pastWeeksNotIncluded=26,
 pThresholdTrend=1,trend=TRUE,
 thresholdMethod="delta",alpha=0.1)
salm1 <- farringtonFlexible(salm,control=control1)
salm2 <- farringtonFlexible(salm,control=control2)

### PLOT THE RESULTS ###
y.max <- max(upperbound(salm1),observed(salm1),upperbound(salm2),na.rm=TRUE)

plot(salm1,ylim=c(0,y.max),
        main='S. Newport in Germany')
lines( 1:(nrow(salm1)+1)-0.5,
       c(upperbound(salm1),upperbound(salm1)[nrow(salm1)]),
      type="s",col='tomato4',lwd=2)
lines( 1:(nrow(salm2)+1)-0.5,
       c(upperbound(salm2),upperbound(salm2)[nrow(salm2)]),
      type="s",col="blueviolet",lwd=2)

legend(c(0,10),c('Alarm','Upperbound with old options','Upperbound with new options'),
      pch=c(24,NA,NA),lty=c(NA,1,1),
        bg="white",lwd=c(2,2,2),col=c('red','tomato4',"blueviolet"))
```

`find.kh` *Determine the k and h values in a standard normal setting*

Description

Given a specification of the average run length in the (a)cceptance and (r)ejected setting determine the k and h values in a standard normal setting.

Usage

```
find.kh(ARLa = 500, ARLr = 7, sided = "one", method = "BFGS", verbose=FALSE)
```

Arguments

`ARLa`	average run length in acceptance setting, aka. in control state. Specifies the number of observations before false alarm.

ARLr	average run length in rejection state, aka. out of control state. Specifies the number of observations before an increase is detected (i.e. detection delay)
sided	one-sided cusum scheme
method	Which method to use in the function optim. Standard choice is BFGS, but in some situation Nelder-Mead can be advantageous.
verbose	gives extra information about the root finding process

Details

Functions from the **spc** package are used in a simple univariate root finding problem.

Value

Returns a list with reference value k and decision interval h.

Examples

```
if (requireNamespace("spc")) {
    find.kh(ARLa=500,ARLr=7,sided="one")
    find.kh(ARLa=500,ARLr=3,sided="one")
}
```

---

findH *Find decision interval for given in-control ARL and reference value*

Description

Function to find a decision interval h* for given reference value k and desired ARL γ so that the average run length for a Poisson or Binomial CUSUM with in-control parameter θ_0, reference value k and is approximately γ, i.e. $\left|\frac{ARL(h^*)-\gamma}{\gamma}\right| < \epsilon$, or larger, i.e. $ARL(h^*) > \gamma$.

Usage

```
findH(ARL0, theta0, s = 1, rel.tol = 0.03, roundK = TRUE,
       distr = c("poisson", "binomial"), digits = 1, FIR = FALSE, ...)

hValues(theta0, ARL0, rel.tol=0.02, s = 1, roundK = TRUE, digits = 1,
      distr = c("poisson", "binomial"), FIR = FALSE, ...)
```

Arguments

ARL0	desired in-control ARL γ
theta0	in-control parameter θ_0
s	change to detect, see details
distr	"poisson" or "binomial"

rel.tol	relative tolerance, i.e. the search for h* is stopped if $\left\lvert\frac{ARL(h^*)-\gamma}{\gamma}\right\rvert <$ rel.tol
digits	the reference value k and the decision interval h are rounded to digits decimal places
roundK	passed to findK
FIR	if TRUE, the decision interval that leads to the desired ARL for a FIR CUSUM with head start $\frac{h}{2}$ is returned
...	further arguments for the distribution function, i.e. number of trials n for binomial cdf

Details

The out-of-control parameter used to determine the reference value k is specified as:

$$\theta_1 = \lambda_0 + s\sqrt{\lambda_0}$$

for a Poisson variate $X \sim Po(\lambda)$

$$\theta_1 = \frac{s\pi_0}{1 + (s-1)\pi_0}$$

for a Binomial variate $X \sim Bin(n, \pi)$

Value

findH returns a vector and hValues returns a matrix with elements

theta0	in-control parameter
h	decision interval
k	reference value
ARL	ARL for a CUSUM with parameters k and h
rel.tol	corresponds to $\left\lvert\frac{ARL(h)-\gamma}{\gamma}\right\rvert$

---

findK *Find reference value*

Description

Calculates the reference value k for a Poisson or binomial CUSUM designed to detect a shift from θ_0 to θ_1

Usage

```
findK(theta0, theta1, distr = c("poisson", "binomial"),
      roundK = FALSE, digits = 1, ...)
```

Arguments

theta0	in-control parameter
theta1	out-of-control parameter
distr	"poisson" or "binomial"
digits	the reference value k is rounded to digits decimal places
roundK	For discrete data and rational reference value there is only a limited set of possible values that the CUSUM can take (and therefore there is also only a limited set of ARLs). If roundK=TRUE, integer multiples of 0.5 are avoided when rounding the reference value k, i.e. the CUSUM can take more values.
...	further arguments for the distribution function, i.e. number of trials n for the binomial cdf.

Value

Returns reference value k.

---

fluBYBW *Influenza in Southern Germany*

Description

Weekly number of influenza A & B cases in the 140 districts of the two Southern German states Bavaria and Baden-Wuerttemberg, for the years 2001 to 2008. These surveillance data have been analyzed originally by Paul and Held (2011) and more recently by Meyer and Held (2014).

Usage

```
data(fluBYBW)
```

Format

An sts object containing 416×140 observations starting from week 1 in 2001.

The population slot contains the population fractions of each district at 31.12.2001, obtained from the Federal Statistical Office of Germany.

The map slot contains an object of class "SpatialPolygonsDataFrame".

Note

Prior to **surveillance** version 1.6-0, data(fluBYBW) contained a redundant last row (417) filled with zeroes only.

Source

Robert Koch-Institut: SurvStat: http://www3.rki.de/SurvStat; Queried on 6 March 2009.

References

Paul, M. and Held, L. (2011) Predictive assessment of a non-linear random effects model for multivariate time series of infectious disease counts. Statistics in Medicine, **30**, 1118-1136.

Meyer, S. and Held, L. (2014): Power-law models for infectious disease spread. *The Annals of Applied Statistics*, **8** (3), 1612-1639.
DOI-Link: http://dx.doi.org/10.1214/14-AOAS743

Examples

```
data("fluBYBW")
# Count time series plot
plot(fluBYBW, type = observed ~ time, legend.opts = NULL)
# Map of disease incidence (per 100000 inhabitants) for the year 2001
plot(fluBYBW, type = observed ~ unit, tps = 1:52, total.args = list(),
     population = fluBYBW@map$X31_12_01 / 100000)
# the overall rate for 2001 shown in the bottom right corner is
sum(observed(fluBYBW[1:52,])) / sum(fluBYBW@map$X31_12_01) * 100000

## Not run:
# Generating an animation takes a while.
# Here we take the first 20 weeks of 2001 (runtime: ~3 minutes).
# The full animation is available in Supplement A of Meyer and Held (2014)
if (require("animation")) {
    oldwd <- setwd(tempdir())  # to not clutter up the current working dir
    saveHTML(animate(fluBYBW, tps = 1:20),
             title="Evolution of influenza in Bayern and Baden-Wuerttemberg",
             ani.width=500, ani.height=600)
    setwd(oldwd)
}

## End(Not run)
```

`formatPval` *Pretty p-Value Formatting*

Description

Just YAPF – yet another p-value formatter...

It is implemented as `sapply(pv, function(p) format.pval(p, digits = if (p<10*eps) 1 else 2, eps = eps)`.

Usage

```
formatPval(pv, eps = 1e-4)
```

Arguments

pv	a numeric vector (of p-values).
eps	a numerical tolerance, see `format.pval`.

Value

The character vector of formatted p-values.

Examples

```
formatPval(c(0.13567, 0.0432, 0.000546, 1e-8))
```

---

glm_epidataCS *Fit an Endemic-Only* twinstim *as a Poisson*-glm

---

Description

An endemic-only twinstim is equivalent to a Poisson regression model for the aggregated number of events, $Y_{[t][s],k}$, by time-space-type cell. The rate of the corresponding Poisson distribution is $e_{[t][s]} \cdot \lambda([t], [s], k)$, where $e_{[t][s]} = |[t]||[s]|$ is a multiplicative offset. Thus, the glm function can be used to fit an endemic-only twinstim. However, wrapping in glm is usually slower.

Usage

```
glm_epidataCS(formula, data, ...)
```

Arguments

formula	an endemic model formula without response, comprising variables of data$stgrid and possibly the variable type for a type-specific model.
data	an object of class "epidataCS".
...	arguments passed to glm. Note that family and offset are fixed internally.

Value

a glm

Author(s)

Sebastian Meyer

Examples

```
data("imdepi")
data("imdepifit")

## Fit an endemic-only twinstim() and an equivalent model wrapped in glm()
fit_twinstim <- update(imdepifit, epidemic = ~0, siaf = NULL,
                       optim.args=list(control=list(trace=0)), verbose=FALSE)
fit_glm <- glm_epidataCS(formula(fit_twinstim)$endemic, imdepi)

## Compare the coefficients
cbind(twinstim=coef(fit_twinstim), glm=coef(fit_glm))
```

```
stopifnot(isTRUE(all.equal(coef(fit_glm), coef(fit_twinstim),
                           tolerance = 0.0005, check.attributes = FALSE)))


### also compare to an equivalent endemic-only hhh4() fit

## first need to aggregate imdepi into an "sts" object
load(system.file("shapes", "districtsD.RData", package="surveillance"))
imdepi_sts <- epidataCS2sts(imdepi, freq=12, start=c(2002,1),
    neighbourhood=NULL, tiles=districtsD, popcol.stgrid="popdensity")

## determine the correct offset to get an equivalent model
offset <- 2 * rep(with(subset(imdepi$stgrid, !duplicated(BLOCK)),
                  stop-start), ncol(imdepi_sts)) *
          sum(districtsD$POPULATION) * population(imdepi_sts)

## fit the model using hhh4()
fit_hhh4 <- hhh4(imdepi_sts, control = list(
    end = list(
        f = addSeason2formula(~I(start/365-3.5), period=365, timevar="start"),
        offset = offset
    ), family = "Poisson", subset = 1:nrow(imdepi_sts),
    data = list(start=with(subset(imdepi$stgrid, !duplicated(BLOCK)), start))))

summary(fit_hhh4)
stopifnot(isTRUE(all.equal(coef(fit_hhh4), coef(fit_glm),
                           check.attributes=FALSE)))
```

---

ha *Hepatitis A in Berlin*

Description

Number of Hepatitis A cases among adult male (age>18) in Berlin 2001-2006. An increase is seen during 2006

Usage

```
data("ha")
data("ha.sts")
```

Format

ha is a disProg object containing 290×12 observations starting from week 1 in 2001 to week 30 in 2006. ha.sts is generated from ha by the converter function disProg2sts using a shape file of Berlin, see the Example given in the help file for class "sts".

Source

Robert Koch-Institut: SurvStat: http://www3.rki.de/SurvStat; Queried on 25 August 2006.

Robert Koch Institut, Epidemiologisches Bulletin 33/2006, p.290.

Examples

```
data(ha)
plot(aggregate(ha))
```

---

hagelloch *1861 Measles Epidemic in the City of Hagelloch, Germany*

Description

Data on the 188 cases in the measles outbreak among children in the German city of Hagelloch (near Tübingen) 1861. The data were originally collected by Dr. Albert Pfeilsticker (1863) and augmented and re-analysed by Dr. Heike Oesterle (1992).

Usage

```
data("hagelloch")
```

Format

Loading `data("hagelloch")` gives two objects: `hagelloch` and `hagelloch.df`. The former is an `"epidata"` object for use with `twinSIR` containing the entire SIR event history of the outbreak in the population of 188 children. The latter is the original `data.frame` of 188 rows with individual information for each infected child.

The covariate information in `hagelloch.df` is as follows:

PN: patient number

NAME: patient name (as a factor)

FN: family index

HN: house number

AGE: age in years

SEX: gender of the individual (factor: male, female)

PRO: `Date` of prodromes

ERU: `Date` of rash

CL: class (factor: preschool, 1st class, 2nd class)

DEAD: `Date` of death (with missings)

IFTO: number of patient who is the putative source of infection (0 = unknown)

SI: serial interval = number of days between dates of prodromes of infection source and infected person

C: complications (factor: no complicatons, bronchopneumonia, severe bronchitis, lobar pneumonia, pseudocroup, cerebral edema)

PR: duration of prodromes in days

CA: number of cases in family

NI: number of initial cases

GE: generation number of the case

TD: day of max. fever (days after rush)

TM: max. fever (degree celsius)

x.loc: x coordinate of house (in meters). Scaling in metres is obtained by multiplying the original coordinates by 2.5 (see details in Neal and Roberts (2004))

y.loc: y coordinate of house (in meters). See also the above description of `x.loc`.

tPRO: Time of prodomes (first symptoms) in days after the start of the epidemic (30 Oct 1861).

tERU: Time upon which the rash first appears.

tDEAD: Time of death, if available.

tR: Time at which the infectious period of the individual is assumed to end. This unknown time is calculated as

$$tR_i = \min tDEAD_i, tERU_i + d_0,$$

where – as in Section 3.1 of Neal and Roberts (2004) – we use $d_0 = 3$.

tI: Time at which the individual is assumed to become infectious. Actually this time is unknown, but we use

$$tI_i = tPRO_i - d_1,$$

where $d_1 = 1$ as in Neal and Roberts (2004).

The time variables describe the transitions of the individual in an Susceptible-Infectious-Revovered (SIR) model. Note that in order to avoid ties in the event times resulting from daily interval censoring, the times have been jittered uniformly within the respective day. The time point 0.5 would correspond to noon of 30 Oct 1861.

The `hagelloch` `"epidata"` object only retains some of the above covariates to save space. Apart from the usual `"epidata"` event columns, `hagelloch` contains a number of extra variables representing distance- and covariate-based weights for the force of infection. These have been computed by specifying `f` and `w` arguments in `as.epidata` at conversion (see the Examples below):

household: the number of currently infectious children in the same household (including the child itself if it is currently infectious), corresponding to `function(u) u == 0` in `f`.

nothousehold: the number of currently infectious children outside the household, corresponding to `function(u) u > 0` in `f`.

c1, c2: the number of children infectious during the respective time block and being members of class 1 and 2, respectively; but the value is 0 if the individual of the row is not herself a member of the respective class. See the Examples below for the corresponding function definitions in `w`.

Source

Thanks to Peter J. Neal, University of Manchester, for providing us with these data, which he again became from Niels Becker, Australian National University. To cite the data, the main references are Pfeilsticker (1863) and Oesterle (1992).

References

Pfeilsticker, A. (1863). Beiträge zur Pathologie der Masern mit besonderer Berücksichtigung der statistischen Verhältnisse, M.D. Thesis, Eberhard-Karls-Universität Tübingen. Available as http://www.archive.org/details/beitrgezurpatho00pfeigoog.

Oesterle, H. (1992). Statistische Reanalyse einer Masernepidemie 1861 in Hagelloch, M.D. Thesis, Eberhard-Karls-Universitäat Tübingen.

Neal, P. J. and Roberts, G. O (2004). Statistical inference and model selection for the 1861 Hagelloch measles epidemic, Biostatistics 5(2):249-261

See Also

twinSIR, epidata

Examples

```
data("hagelloch")

head(hagelloch.df)   # original data documented in Oesterle (1992)
head(as.data.frame(hagelloch))   # derived "epidata" object


### How the "epidata" 'hagelloch' was created from 'hagelloch.df'

stopifnot(all.equal(hagelloch,
  as.epidata(
    hagelloch.df, t0 = 0, tI.col = "tI", tR.col = "tR",
    id.col = "PN", coords.cols = c("x.loc", "y.loc"),
    f = list(
        household    = function(u) u == 0,
        nothousehold = function(u) u > 0
    ),
    w = list(
        c1 = function (CL.i, CL.j) CL.i == "1st class" & CL.j == CL.i,
        c2 = function (CL.i, CL.j) CL.i == "2nd class" & CL.j == CL.i
    ),
    keep.cols = c("SEX", "AGE", "CL"))
))


### Basic plots produced from hagelloch.df

# Show case locations as in Neal & Roberts (different scaling) using
# the data.frame (promoted to a SpatialPointsDataFrame)
coordinates(hagelloch.df) <- c("x.loc","y.loc")
```

```
plot(hagelloch.df, xlab="x [m]", ylab="x [m]", pch=15, axes=TRUE,
     cex=sqrt(multiplicity(hagelloch.df)))

# Epicurve
hist(as.numeric(hagelloch.df$tI), xlab="Time (days)", ylab="Cases", main="")


### SIR model information for population & individuals

(s <- summary(hagelloch))
plot(s, col=c("green","red","darkgray"))
stateplot(s, id=c("187"))

## Not run:
  # Show a dynamic illustration of the spread of the infection
  animate(hagelloch,time.spacing=0.1,legend.opts=list(x="topleft"),sleep=1/100)

## End(Not run)
```

`hepatitisA` *Hepatitis A in Germany*

Description

Weekly number of reported hepatitis A infections in Germany 2001-2004.

Usage

```
data(hepatitisA)
```

Format

A `disProg` object containing 208×1 observations starting from week 1 in 2001 to week 52 in 2004.

Source

Robert Koch-Institut: SurvStat: http://www3.rki.de/SurvStat; Queried on 11 01 2005.

Examples

```
data(hepatitisA)
plot(hepatitisA)
```

hhh4 *Fitting HHH Models with Random Effects and Neighbourhood Structure*

Description

Fits a Poisson or negative binomial model with conditional mean

$$\mu_{it} = \lambda_{it} y_{i,t-1} + \phi_{it} \sum_{j \neq i} w_{ji} y_{j,t-1} + e_{it} \nu_{it}$$

containing epidemic and endemic components to a multivariate time series of counts Y_{it} (unit i, period t). In the case of a negative binomial model, the conditional variance is $\mu_{it}(1 + \psi \mu_{it})$ with overdispersion parameter ψ. The three unknown quantities of the mean μ_{it},

- λ_{it} in the autoregressive (`ar`) component,
- ϕ_{it} in the neighbor-driven (`ne`) component, and
- ν_{it} in the endemic (`end`) component,

are log-linear predictors incorporating time-/unit-specific covariates. They may contain unit-specific random intercepts (`ri`) as proposed by Paul and Held (2011). The e_{it} is a (multiplicative) endemic offset; it is also possible to include such offsets in the epidemic components. The w_{ji} are neighbourhood weights, which can either be prespecified or estimated parametrically as proposed by Meyer and Held (2014).

Usage

```
hhh4(stsObj,
     control = list(
               ar = list(f = ~ -1,
                         offset = 1,
                         lag = 1,
                         weights = NULL,
                         initial = NULL
                         ),
               ne = list(f = ~ -1,
                         offset = 1,
                         lag = 1,
                         weights = neighbourhood(stsObj) == 1,
                         initial = NULL
                         ),
               end = list(f = ~ 1,
                          offset = 1,
                          initial = NULL
                          ),
               family = c("Poisson","NegBin1","NegBinM"),
               subset = 2:nrow(stsObj),
```

```
                optimizer = list(stop = list(tol=1e-5, niter=100),
                                 regression = list(method="nlminb"),
                                 variance = list(method="nlminb")),
                verbose = FALSE,
                start = list(fixed=NULL, random=NULL, sd.corr=NULL),
                data = list(t = epoch(stsObj)-1),
                keep.terms = FALSE
                ),
    check.analyticals = FALSE
)
```

Arguments

`stsObj` object of class "`sts`" containing the multivariate count data time series

`control` a list containing the model specification and control arguments:

`ar` Model for the autoregressive component given as list with the following components:

f = ~ -1 a formula specifying $\log(\lambda_{it})$

offset = 1 optional multiplicative offset, either 1 or a matrix of the same dimension as `observed(stsObj)`

lag = 1 a positive integer meaning autoregression on $y_{i,t-lag}$

weights = NULL optional weights, only used if model is a contact matrix (currently not implemented)

initial = NULL vector with initial values for parameters if `f = ~1` (not really used ATM)

`ne` Model for the neighbor-driven component given as list with the following components:

f = ~ -1 a formula specifying $\log(\phi_{it})$

offset = 1 optional multiplicative offset, either 1 or a matrix of the same dimension as `observed(stsObj)`

lag = 1 a non-negative integer meaning dependency on $y_{j,t-lag}$

weights = neighbourhood(stsObj) == 1 neighbourhood weights w_{ji}. The default corresponds to the original formulation by Held et al (2005), i.e., the spatio-temporal component incorporates an unweighted sum over the lagged cases of the first-order neighbours. See Paul et al (2008) and Meyer and Held (2014) for alternative specifications, e.g., `W_powerlaw`. Time-varying weights are possible by specifying an array of `dim()` `c(nUnits, nUnits, nTime)`, where `nUnits=ncol(stsObj)` and `nTime=nrow(stsObj)`.

initial = NULL vector with initial values for parameter if `f = ~1` (not really used ATM)

`end` Model for the endemic component given as list with the following components

f = ~ 1 a formula specifying $\log(\nu_{it})$

offset = 1 optional multiplicative offset e_{it}, either 1 or a matrix of the same dimension as `observed(stsObj)`

initial = NULL vector with initial values for parameter if `f = ~1` (not really used ATM)

`family` Distributional family – either `"Poisson"`, or the Negative Binomial distribution with one common overdispersion parameter for all units (`"NegBin1"`) or an overdispersion parameter for each unit (`"NegBinM"`).

`subset` Typically `2:nrow(obs)` if model contains autoregression

`optimizer` a list of three lists of control arguments.
The `"stop"` list specifies two criteria for the outer optimization of regression and variance parameters: the relative `tol`erance for parameter change using the criterion `max(abs(x[i+1]-x[i])) / max(abs(x[i]))`, and the maximum number `niter` of outer iterations.
Control arguments for the single optimizers are specified in the lists named `"regression"` and `"variance"`. `method="nlminb"` is the default optimizer for both (taking advantage of the analytical Fisher information matrices), however, the `method`s from `optim` may also be specified (as well as `"nlm"` but that one is not recommended here). Especially for the variance updates, Nelder-Mead optimization (`method="Nelder-Mead"`) is an attractive alternative. All other elements of these two lists are passed as `control` arguments to the chosen `method`, e.g., if `method="nlminb"` adding `iter.max=50` increases the maximum number of inner iterations from 20 (default) to 50.

`verbose` non-negative integer (usually in the range `0:3`) specifying the amount of tracing information to be output during optimization.

`start` a list of initial parameter values; overrides any initial values in formulas (this is currently the only way to specify initial values).

`data` a named list of covariates that are to be included as fixed effects (see `fe`) in any of the 3 component formulae. By default, the time variable `t` is available and used for seasonal effects created by `addSeason2formula`. In general, covariates in this list can be either vectors of length `nrow(stsObj)` interpreted as time-varying but common across all units, or matrices of the same dimension as the disease counts `observed(stsObj)`.

`keep.terms` logical indicating if the terms object used in the fit is to be kept as part of the returned object. This is usually not necessary, since the terms object is reconstructed by the `terms`-method for class `"hhh4"` if necessary (based on `stsObj` and `control`, which are both part of the returned `"hhh4"` object).

`check.analyticals`
logical (or a subset of `c("numDeriv", "maxLik")`), indicating if (how) the implemented analytical score vector and Fisher information matrix should be checked against numerical derivatives at the parameter starting values, using the packages **numDeriv** and/or **maxLik**. If activated, `hhh4` will return a list containing the analytical and numerical derivatives for comparison (no ML estimation will be performed). This is mainly intended for internal use by the package developers.

Details

For further details see `vignette("hhh4")` and the references.

Value

hhh4 returns an object of class "hhh4", which is a list containing the following components:

coefficients	named vector with estimated (regression) parameters of the model
se	estimated standard errors (for regression parameters)
cov	covariance matrix (for regression parameters)
Sigma	estimated variance-covariance matrix of random effects
Sigma.orig	estimated variance parameters on internal scale used for optimization
Sigma.cov	inverse of marginal Fisher information (on internal scale), i.e., the asymptotic covariance matrix of Sigma.orig
call	the matched call
dim	vector with number of fixed and random effects in the model
loglikelihood	(penalized) loglikelihood evaluated at the MLE
margll	(approximate) log marginal likelihood should the model contain random effects
convergence	logical. Did optimizer converge?
fitted.values	fitted mean values $\mu_{i,t}$
control	control object of the fit
terms	the terms object used in the fit if keep.terms = TRUE and NULL otherwise
stsObj	the supplied stsObj
lags	named integer vector of length two containing the lags used for the epidemic components "ar" and "ne", respectively. The corresponding lag is NA if the component was not included in the model.
nObs	number of observations used for fitting the model
nTime	number of time points used for fitting the model
nUnit	number of units (e.g. areas) used for fitting the model
runtime	the proc.time-queried time taken to fit the model, i.e., a named numeric vector of length 5 of class "proc_time"

Author(s)

M. Paul, S. Meyer, and L. Held

References

Held, L., Höhle, M., Hofmann, M. (2005): A statistical framework for the analysis of multivariate infectious disease surveillance counts. *Statistical Modelling*, **5**, 187–199.

Paul, M., Held, L. and Toschke, A. M. (2008): Multivariate modelling of infectious disease surveillance data. *Statistics in Medicine*, **27**, 6250–6267.

Paul, M. and Held, L. (2011): Predictive assessment of a non-linear random effects model for multivariate time series of infectious disease counts. *Statistics in Medicine*, **30**, 1118–1136

Held, L. and Paul, M. (2012): Modeling seasonality in space-time infectious disease surveillance data. *Biometrical Journal*, **54**, 824–843

Meyer, S. and Held, L. (2014): Power-law models for infectious disease spread. *The Annals of Applied Statistics*, **8** (3), 1612-1639.
DOI-Link: http://dx.doi.org/10.1214/14-AOAS743

See Also

algo.hhh, fe, ri

Examples

```
######################################################################
# Fit some models from ?algo.hhh
######################################################################

## univariate salmonella agona data
data(salmonella.agona)
# convert to sts class
salmonella <- disProg2sts(salmonella.agona)

# generate formula for temporal and seasonal trends
f.end <- addSeason2formula(f = ~ 1 + t, S=1, period=52)
model1 <- list(ar = list(f = ~ 1), end = list(f =f.end),
               family = "NegBin1")
# run model
res <- hhh4(salmonella, model1)
summary(res, idx2Exp=1, amplitudeShift=TRUE)


## multivariate time series:
# measles cases in Lower Saxony, Germany
data(measles.weser)
measles <- disProg2sts(measles.weser)

# same model as above
summary(hhh4(measles, control=model1))

# now use region-specific intercepts in endemic component
f.end2 <- addSeason2formula(f = ~ -1 + fe(1, which=rep(TRUE, ncol(measles))) + t,
                            S = 1, period = 52)
model2 <- list(ar = list(f = ~ 1),
               end = list(f = f.end2, offset = population(measles)),
               family = "NegBin1")
# run model
summary(hhh4(measles, control=model2), idx2Exp=1, amplitudeShift=TRUE)

# include autoregressive parameter phi for adjacent "Kreise"
# no linear trend in endemic component
f.end3 <- addSeason2formula(f = ~ -1 + fe(1, which=rep(TRUE, ncol(measles))),
                            S = 1, period = 52)
model3 <- list(ar = list(f = ~ 1),
               ne = list(f = ~1),
               end = list(f = f.end3, offset= population(measles)),
               family = "NegBin1")
# run model
res3 <- hhh4(measles, control=model3)
summary(res3, idx2Exp=1:2, amplitudeShift=TRUE)
```

```
## Not run:
######################################################################
# Fit the models from the Paul & Held (2011) paper for the influenza data
# from Bavaria and Baden-Wuerttemberg (this takes some time!)
# For further documentation see also the vignette.
######################################################################

data("fluBYBW")

############################################################
## generate formula for temporal and seasonal trends
f.end <- addSeason2formula(f = ~ -1 + ri(type="iid", corr="all") +
                               I((t-208)/100), S=3, period=52)

## details for optimizer
opt <- list(stop = list(tol=1e-5, niter=200),
            regression = list(method="nlminb"),
            variance = list(method="nlminb"))

###########################
## models
# A0
cntrl_A0 <- list(ar = list(f = ~ -1),
                 end = list(f =f.end, offset = population(fluBYBW)),
                 family = "NegBin1", optimizer = opt, verbose = 1)
summary(res_A0 <- hhh4(fluBYBW,cntrl_A0))

# B0
cntrl_B0 <- list(ar = list(f = ~ 1),
                 end = list(f =f.end, offset = population(fluBYBW)),
                 family = "NegBin1", optimizer = opt, verbose=1)
res_B0 <- hhh4(fluBYBW,cntrl_B0)


# C0
cntrl_C0 <- list(ar = list(f = ~ -1 + ri(type="iid", corr="all")),
                 end = list(f =f.end, offset = population(fluBYBW)),
                 family = "NegBin1", optimizer = opt, verbose=1)
res_C0 <- hhh4(fluBYBW,cntrl_C0)


#A1

# weight matrix w_ji = 1/(No. neighbors of j) if j ~ i, and 0 otherwise
wji <- neighbourhood(fluBYBW)/rowSums(neighbourhood(fluBYBW))

cntrl_A1 <- list(ar = list(f = ~ -1),
                 ne = list(f = ~ 1, weights = wji),
                 end = list(f =f.end, offset = population(fluBYBW)),
                 family = "NegBin1", optimizer = opt, verbose=1)
res_A1 <- hhh4(fluBYBW,cntrl_A1)
```

```
# B1
cntrl_B1 <- list(ar = list(f = ~ 1),
                 ne = list(f = ~ 1, weights = wji),
                 end = list(f =f.end, offset = population(fluBYBW)),
                 family = "NegBin1", optimizer = opt, verbose=1)
res_B1 <- hhh4(fluBYBW,cntrl_B1)


# C1
cntrl_C1 <- list(ar = list(f = ~ -1 + ri(type="iid", corr="all")),
                 ne = list(f = ~ 1, weights = wji),
                 end = list(f =f.end, offset = population(fluBYBW)),
                 family = "NegBin1", optimizer = opt, verbose=1)
res_C1 <- hhh4(fluBYBW,cntrl_C1)


#A2
cntrl_A2 <- list(ar = list(f = ~ -1),
                 ne = list(f = ~ -1 + ri(type="iid",corr="all"), weights=wji),
                 end = list(f =f.end, offset = population(fluBYBW)),
                 family = "NegBin1", optimizer = opt, verbose=1)
res_A2 <- hhh4(fluBYBW,cntrl_A2)


# B2
cntrl_B2 <- list(ar = list(f = ~ 1),
                 ne = list(f = ~ -1 + ri(type="iid",corr="all"), weights =wji),
                 end = list(f =f.end, offset = population(fluBYBW)),
                 family = "NegBin1", optimizer = opt, verbose=1)
res_B2 <- hhh4(fluBYBW,cntrl_B2)

# C2
cntrl_C2 <- list(ar = list(f = ~ -1 + ri(type="iid", corr="all")),
                 ne = list(f = ~ -1 + ri(type="iid",corr="all"), weights =wji),
                 end = list(f =f.end, offset = population(fluBYBW)),
                 family = "NegBin1", optimizer = opt, verbose=1,
                 start=list(fixed=fixef(res_B0),random=c(rep(0,140),
                         ranef(res_B0)), sd.corr=c(-.5,res_B0$Sigma.orig,0)))
res_C2 <- hhh4(fluBYBW,cntrl_C2)


# D
cntrl_D <- list(ar = list(f = ~ 1),
                ne = list(f = ~ -1 + ri(type="iid"), weights = wji),
                end = list(f =addSeason2formula(f = ~ -1 + ri(type="car") +
                                          I((t-208)/100), S=3, period=52),
                           offset = population(fluBYBW)),
                family = "NegBin1", optimizer = opt, verbose=1)
res_D <- hhh4(fluBYBW,cntrl_D)
```

```
## End(Not run)
```

---

hhh4_formula *Specify Formulae in a Random Effects HHH Model*

Description

The special functions `fe` and `ri` are used to specify (unit-specific) effects of covariates and a random intercept term, respectively, in formulae used in the function `hhh4`.

Usage

```
fe(x, which = NULL, initial = NULL)

ri(type = c("iid","car")[1], corr = c("none", "all")[1],
   initial.var = NULL, initial.re = NULL)
```

Arguments

`x`	an expression like `sin(2*pi*t/52)` involving the time variable `t`, or just `1` for an intercept. In general this covariate expression might use any variables contained in the `control$data` argument of the parent `hhh4` call.
`which`	vector of logicals indicating which unit should get an unit-specific parameter. If `NULL`, the effect of the covariate is assumbed to be the same for all units.
`initial`	initial values (on internal scale!) for the fixed effects used for optimization. Not really used ATM.
`type`	random intercepts either follow an IID or a CAR model
`corr`	logical switch indicating whether random effects in different components (such as `ar` and `end`) should be correlated or not.
`initial.var`	initial values (on internal scale!) for the variance components used for optimization. Not really used ATM.
`initial.re`	initial values (on internal scale!) for the random effects used for optimization. Not really used ATM.

Note

This function should only be used in formula objects for `hhh4`, and is not intended for direct calling.

If unit-specific or random intercepts are specified, an overall intercept must be excluded with `-1`.

See Also

`addSeason2formula`, usage of formulae in the vignette and in examples of `hhh4`.

Examples

```
## Not run:
# some calls of the fitting function 'hhh4':
# see vignette("hhh4") for further details

data("influMen")
fluMen <- disProg2sts(influMen)
meningo <- fluMen[, "meningococcus"]

## Ex: univariate time series of meningococcal infections in Germany
# Negative binomial model with
# endemic component: Intercept + S = 1 sine/cosine pair
# autoregressive component: Intercept

f.S1 <- addSeason2formula(f = ~ 1, S = 1, period = 52)
hhh4(meningo, control = list(ar = list(f = ~ 1),
                             end = list(f = f.S1),
                             family = "NegBin1"))


## Ex: disease-specific intercept in influenza/meningococcal time series
# Negative binomial model with
# autoregressive component: disease-specific intercepts
# neighbour-driven component: only intercept for flu -> men
# endemic component: S=3 and S=1 sine/cosine pairs for flu and men, respectively

neighbourhood(fluMen)["meningococcus","influenza"] <- 0
f.end <- addSeason2formula(f = ~ -1 + fe(1, which = c(TRUE, TRUE)),
                           S = c(3, 1),
                           period = 52)
m <- list(ar = list(f = ~ -1 + fe(1, which = c(TRUE, TRUE))),
          ne = list(f = ~ -1 + fe(1, which = c(FALSE, TRUE))),
          end = list(f = f.end),
          family = "NegBinM"
          )
hhh4(fluMen, control = m)


## Ex: (correlated) random intercepts for influenza in Southern Germany
# Negative binomial model with
# autoregressive component: Intercept
# neighbour-driven component: random intercepts
# endemic component: random intercepts + trend + S = 3 sine/cosine pairs

data("fluBYBW")
f.end <- addSeason2formula(f = ~ -1 + ri(type = "iid", corr="all") +
                                  I((t-208)/100), S = 3, period = 52)
wji <- neighbourhood(fluBYBW)/rowSums(neighbourhood(fluBYBW))
model.B2 <- list(ar = list(f = ~ 1),
                 ne = list(f = ~ -1 + ri(type = "iid", corr="all"),
                           weights = wji),
                 end = list(f = f.end, offset = population(fluBYBW)),
```

```
                family = "NegBin1")
hhh4(fluBYBW, model.B2)


## Ex: measles in Germany (see vignette("hhh4"))
# Poisson model with
# autoregressive component: Intercept + vaccination coverage info
# endemic component: Intercept + S = 1 sine/cosine pair

f.end <- addSeason2formula(f = ~ 1, S = 1, period = 26)
model.A0 <- list(ar = list(f = ~ 1 + logVac0),
                 end = list(f = f.end, offset = population(measles2w)),
                 data = list(logVac0 = log(vac0)))

## End(Not run)
```

---

hhh4_methods *Print, Summary and other Standard Methods for "hhh4" Objects*

Description

Besides `print` and `summary` methods there are also some standard extraction methods defined for objects of class `"hhh4"` resulting from a call to hhh4.

Usage

```
## S3 method for class 'hhh4'
print(x, digits = max(3, getOption("digits") - 3), ...)
## S3 method for class 'hhh4'
summary(object, maxEV = FALSE, ...)

## S3 method for class 'hhh4'
coef(object, se = FALSE, reparamPsi = TRUE,
     idx2Exp = NULL, amplitudeShift = FALSE, ...)
## S3 method for class 'hhh4'
fixef(object, ...)
## S3 method for class 'hhh4'
ranef(object, tomatrix = FALSE, ...)

## S3 method for class 'hhh4'
formula(x, ...)
## S3 method for class 'hhh4'
nobs(object, ...)
## S3 method for class 'hhh4'
logLik(object, ...)

## S3 method for class 'hhh4'
vcov(object, reparamPsi = TRUE,
```

```
     idx2Exp = NULL, amplitudeShift = FALSE, ...)
## S3 method for class 'hhh4'
confint(object, parm, level = 0.95,
        reparamPsi = TRUE, idx2Exp = NULL, amplitudeShift = FALSE, ...)
```

Arguments

x, object	an object of class "hhh4".
digits	the number of significant digits to use when printing
maxEV	logical indicating if the summary should contain the (range of the) dominant eigenvalue as a measure of the importance of the epidemic components. By default, the value is not calculated as this may take some seconds depending on the number of time points and units in object$stsObj.
...	For the print, summary, fixef and ranef methods: arguments passed to coef. For the remaining methods: unused (argument of the generic).
reparamPsi	logical. If TRUE (default), the overdispersion parameter from the negative binomial distribution is transformed from internal scale (-log) to standard scale, where zero corresponds to a Poisson distribution.
se	logical switch indicating if standard errors are required
idx2Exp	vector with integers idicating the parameters which should be returned on exp-scale.
amplitudeShift	logical switch indicating whether the parameters for sine/cosine terms modelling seasonal patterns (see addSeason2formula) should be transformed to an amplitude/shift formulation.
tomatrix	logical. If FALSE (default), the vector of all random effects is returned (as used internally). However, for random intercepts of type="car", the number of parameters is one less than the number of regions and the individual parameters are not obviously linked to specific regions. Setting tomatrix to TRUE returns a more useful representation of random effects in a matrix with as many rows as there are regions and as many columns as there are random effects. Here, any CAR-effects are transformed to region-specific effects.
parm	a vector of numbers or names, specifying which parameters are to be given confidence intervals. If missing, all parameters are considered.
level	the confidence level required.

Value

The coef-method returns all estimated (regression) parameters from a hhh4 model as proposed by Paul and Held (2011). If the model includes random effects, those can be extracted with ranef, whereas fixef returns the fixed parameters.

The formula-method returns the formulae used for the three log-linear predictors in a list with elements "ar", "ne", and "end". The nobs-method returns the number of observations used for model fitting. The logLik-method returns an object of class "logLik" with "df" and "nobs" attributes. For a random effects model, the value of the *penalized* log-likelihood at the MLE is returned, but degrees of freedom are not available (NA_real_). As a consequence, AIC and BIC are only well defined for models without random effects; otherwise these functions return NA_real_.

The vcov-method returns the estimated variance-covariance matrix of the *regression* parameters. The estimated variance-covariance matrix of random effects is available as `object$Sigma`. The confint-method returns Wald-type confidence intervals (assuming asymptotic normality).

Author(s)

Michaela Paul and Sebastian Meyer

References

Paul, M. and Held, L. (2011) Predictive assessment of a non-linear random effects model for multivariate time series of infectious disease counts. Statistics in Medicine, **30**, 1118–1136.

See Also

the plot and update methods for fitted `"hhh4"` models.

---

hhh4_predict *Predictions from a* hhh4 *Model*

Description

Get fitted (component) means from a hhh4 model.

Usage

```
## S3 method for class 'hhh4'
predict(object, newSubset=object$control$subset,
        type="response", ...)
```

Arguments

`object`	fitted hhh4 model (class `"hhh4"`).
`newSubset`	subset of time points for which to return the predictions. Defaults to the subset used for fitting the model, and must be a subset of `1:nrow(object$stsObj)`.
`type`	the type of prediction required. The default (`"response"` or, equivalently, `"mean"`) is on the scale of the response variable (mean = endemic plus epidemic components). The alternatives are: `"endemic"`, `"epidemic"`, `"epi.own"` (i.e. the autoregresssive part), and `"epi.neighbours"` (i.e. the spatio-temporal part).
`...`	unused (argument of the generic).

Value

matrix of fitted means for each time point (of `newSubset`) and region.

Author(s)

Michaela Paul and Sebastian Meyer

hhh4_simulate *Simulates data based on the model proposed by Paul and Held (2011)*

Description

Simulates a multivariate time series of counts based on the Poisson/Negative Binomial model as described in Paul and Held (2011).

Usage

```
## S3 method for class 'hhh4'
simulate(object, nsim = 1, seed = NULL, y.start = NULL,
         subset = 1:nrow(object$stsObj), coefs = coef(object),
         components = c("ar","ne","end"), simplify = nsim>1, ...)
```

Arguments

object	an object of class "hhh4".
nsim	number of time series to simulate. Defaults to 1.
seed	an object specifying how the random number generator should be initialized for simulation (via set.seed). The initial state will also be stored as an attribute "seed" of the result. The original state of the .Random.seed will be restored at the end of the simulation. By default (NULL), neither initialization nor recovery will be done. This behaviour is copied from the simulate.lm method.
y.start	vector or matrix (with ncol(object$stsObj) columns) with starting counts for the epidemic components. If NULL, the observed means in the respective units of the data in object during subset are used.
subset	time period in which to simulate data. Defaults to the whole period.
coefs	coefficients used for simulation from the model in object. Default is to use the fitted parameters. Note that the coefs-vector must be in the same order and scaling as coef(object).
components	character vector indicating which components of the fitted model object should be active during simulation. For instance, a simulation with components="end" is solely based on the fitted endemic mean.
simplify	logical indicating if only the simulated counts (TRUE) or the full "sts" object (FALSE) should be returned for every replicate. By default a full "sts" object is returned iff nsim=1.
...	unused (argument of the generic).

Details

Simulates data from a Poisson or a Negative Binomial model with mean

$$\mu_{it} = \lambda_{it} y_{i,t-1} + \phi_{it} \sum_{j \neq i} w_{ji} y_{j,t-1} + \nu_{it}$$

where $\lambda_{it} > 0$, $\phi_{it} > 0$, and $\nu_{it} > 0$ are parameters which are modelled parametrically. The function uses the model and parameter estimates of the fitted `object` to simulate the time series.

With the argument `coefs` it is possible to simulate from the model as specified in `object`, but with different parameter values.

Value

If `simplify=FALSE`: an object of class "`sts`" (`nsim = 1`) or a list of those (`nsim > 1`).

If `simplify=TRUE`: an array of dimension `c(length(subset), ncol(object$stsObj), nsim)`, where the third dimension is dropped if `nsim=1` (yielding a matrix).

Author(s)

Michaela Paul and Sebastian Meyer

References

Paul, M. and Held, L. (2011) Predictive assessment of a non-linear random effects model for multivariate time series of infectious disease counts. Statistics in Medicine, **30**, 1118–1136

See Also

`hhh4`, `simHHH`

Examples

```
data(influMen)
# convert to sts class and extract meningococcal disease time series
meningo <- disProg2sts(influMen)[,2]

# fit model
fit <- hhh4(meningo, control = list(ar = list(f = ~ 1),
            end = list(f = addSeason2formula(S = 1, period = 52)),
            family = "NegBin1"))
plot(fit)

# simulate from model
simData <- simulate(fit, seed=1234)

# plot simulated data
plot(simData, main = "simulated data", legend.opts = NULL, xaxis.labelFormat=NULL)

# consider a Poisson instead of a NegBin model
coefs <- coef(fit)
coefs["overdisp"] <- 0

simData2 <- simulate(fit, seed=123, coefs = coefs)
plot(simData2, main = "simulated data: Poisson model",
     legend.opts = NULL, xaxis.labelFormat = NULL)

# consider a model with higher autoregressive parameter
```

```
coefs <- coef(fit)
coefs[1] <- log(0.5)

simData3 <- simulate(fit, seed=321, coefs = coefs)
plot(simData3, main = "simulated data: lambda = 0.5",
     legend.opts = NULL, xaxis.labelFormat = NULL)
```

---

hhh4_update update *a fitted* "hhh4" *model*

Description

Re-fit a "hhh4" model with a modified control list.

Usage

```
## S3 method for class 'hhh4'
update(object, ..., S = NULL, subset.upper = NULL,
       use.estimates = FALSE, evaluate = TRUE)
```

Arguments

object	a fitted "hhh4" model.
...	components modifying the original control list for hhh4. Modifications are performed by modifyList(object$control, list(...)).
S	a named list of numeric vectors serving as argument for addSeason2formula, or NULL (meaning no modification of seasonal terms). This argument provides a convenient way of changing the number of harmonics in the formulae of the model components "ar", "ne" and "end" (to be used as names of the list). Non-specified components are not touched. Updating the i'th component's formula works by first dropping all sine and cosine terms and then applying addSeason2formula with arguments S = S[[i]] and period = object$stsObj@freq. Note that this step of updating seasonality is processed after modification of the control list by the ... arguments.
subset.upper	if a scalar value, restrict the new fit to those epochs of object$control$subset which are <= subset.upper. This argument is used by oneStepAhead.
use.estimates	logical specifying if the coef(object) should be used as starting values for the new fit. This currently only works if the set of parameters is unchanged by the updated model formulation and thus defaults to FALSE.
evaluate	logical indicating if the updated model should be fitted directly (defaults to TRUE). Otherwise, the updated control list is returned.

Value

If evaluate = TRUE the re-fitted object, otherwise the updated control list for hhh4.

Author(s)

Sebastian Meyer

See Also

hhh4

Examples

```
data("salmonella.agona")
## convert to sts class
salmonella <- disProg2sts(salmonella.agona)

## fit a basic model
fit0 <- hhh4(salmonella,
             list(ar = list(f = ~1), end = list(f = addSeason2formula(~t))))

## update: Poisson -> NegBin1, component seasonality
fit1 <- update(fit0, family = "NegBin1", S = list(end=2, ar=2))

## compare fits
AIC(fit0, fit1)
opar <- par(mfrow=c(2,2))
plot(fit0, type="fitted", names="fit0", par.settings=NULL)
plot(fit1, type="fitted", names="fit1", par.settings=NULL)
plot(fit0, fit1, type="season", components=c("end", "ar"), par.settings=NULL)
par(opar)
```

---

hhh4_validation *Predictive Model Assessment for* hhh4 *Models*

Description

The function oneStepAhead computes successive one-step-ahead predictions for a (random effects) HHH model fitted by hhh4.

The function scores computes a number of (strictly) proper scoring rules based on such one-step-ahead predictions.

See Paul and Held (2011) for further details.

Usage

```
oneStepAhead(result, tp, type = c("rolling", "first", "final"),
             which.start = c("current", "final"),
             keep.estimates = FALSE, verbose = TRUE, cores = 1)

scores(object, which = c("logs", "rps", "dss", "ses"), units = NULL,
       sign = FALSE, individual = FALSE)
```

Arguments

result	fitted hhh4 model (class "hhh4").
tp	numeric vector of length 1 or 2. One-step-ahead predictions are computed from time points tp[1], ..., tp[2] (yielding predictions for time points tp[1]+1, ...), where tp[2] defaults to the penultimate time point of result$control$subset.
type	The default "rolling" procedure sequentially refits the model up to each time point in tp and computes the one-step-ahead predictions for the respective next time point. The alternative types are no true one-step-ahead predictions but much faster: "first" will refit the model for the first time point tp[1] only and use this specific fit to calculate all subsequent predictions, whereas "final" will just use result to calculate these. The latter case thus gives nothing else than a subset of result$fitted.values, if the tp's are part of the fitted subset result$control$subset.
which.start	Which initial values should be used when successively refitting the model for subsets of the data (up to time point tp[1], up to tp[1]+1, ...) if type="rolling"? Default ("current") is to use the fitted parameters from the previous time point. Alternatively, "final" means to always use the fitted values from result as initial values for the model update. which.start is ignored for other types.
keep.estimates	logical indicating if parameter estimates and log-likelihoods from the successive fits should be returned.
verbose	non-negative integer (usually in the range 0:3) specifying the amount of tracing information to output. During hhh4 model updates, the following verbosity is used: 0 if cores > 1, otherwise verbose-1 if there is more than one time point to predict, otherwise verbose.
cores	the number of cores to use when computing the predictions for the set of time points tp in parallel (with mclapply). Note that parallelization is not possible in the default setting type="rolling" and which.start="current" (use which.start="final" for this to work).
object	result of oneStepAhead.
which	character vector determining which scores to compute. The package **surveillance** implements the following proper scoring rules: logarithmic score ("logs"), ranked probability score ("rps"), Dawid-Sebastiani score ("dss"), and squared error score ("ses"). The normalized SES ("nses") is also available but it is improper and not computed by default. It is possible to name own scoring rules in which. These must be functions of (x, mu, size), where all arguments are time x unit matrices, except that size is NULL in case of a Poisson model. See the package-internal scoring rules for guidance, e.g., surveillance:::logs or surveillance:::dss.
units	integer or character vector indexing the units for which the scores should be computed. By default (NULL) all units are considered.
sign	logical indicating if the function should also return sign(x-mu), i.e., the sign of the difference between the observed counts and corresponding predictions. This does not really make sense when averaging over multiple units with individual=FALSE.
individual	logical indicating if the individual scores of the units should be returned. By default (FALSE), the individual scores are averaged over all units.

Value

oneStepAhead returns a list with the following components:

pred	one-step-ahead predictions in a matrix, where each row corresponds to one of the time points requested via the argument tp, and which has ncol(result$stsObj) unit-specific columns. The rownames indicate the predicted time points and the column names are identical to colnames(result$stsObj).
observed	matrix with observed counts at the predicted time points. It has the same dimensions and names as pred.
psi	in case of a negative-binomial model, a matrix of the estimated overdispersion parameter(s) at each time point on the internal -log-scale (1 column if "NegBin1", ncol(observed) columns if "NegBinM"). For a "Poisson" model, this component is NULL.
allConverged	logical indicating if all successive fits converged.

If keep.estimates=TRUE, there are the following additional elements:

coefficients	matrix of estimated regression parameters from the successive fits.
Sigma.orig	matrix of estimated variance parameters from the successive fits.
logliks	matrix with columns "loglikelihood" and "margll" with their obvious meanings.

The function scores computes the scoring rules specified in the argument which. If multiple units are selected and individual=TRUE, the result is an array of dimensions c(nrow(pred),length(units),5+sign) (up to **surveillance** 1.8-0, the first two dimensions were collapsed to give a matrix). Otherwise, the result is a matrix with nrow(pred) rows and 5+sign columns. If there is only one predicted time point, the first dimension is dropped in both cases. CAVE: The order of the rows (time points) is reversed!

Author(s)

Michaela Paul and Sebastian Meyer

References

Czado, C., Gneiting, T. and Held, L. (2009) Predictive model assessment for count data. Biometrics, **65**, 1254–1261

Paul, M. and Held, L. (2011) Predictive assessment of a non-linear random effects model for multivariate time series of infectious disease counts. Statistics in Medicine, **30**, 1118–1136

See Also

vignette("hhh4") and hhh4.

Examples

```
### univariate salmonella agona data

data("salmonella.agona")
## convert to sts class
salmonella <- disProg2sts(salmonella.agona)

## generate formula for temporal and seasonal trends
f.end <- addSeason2formula(f = ~1 + t, S=1, period=52)
model1 <- list(ar = list(f=~1), end = list(f=f.end), family = "NegBin1")
## fit the model
result <- hhh4(salmonella, model1)

## do sequential one-step-ahead predictions for the last 5 weeks
pred <- oneStepAhead(result, nrow(salmonella)-5, type="rolling",
                     which.start="final", verbose=FALSE)



## oneStepAhead(..., type="final") just means fitted values
stopifnot(identical(
    unname(oneStepAhead(result, nrow(salmonella)-5,
                        type="final", verbose=FALSE)$pred),
    unname(tail(fitted(result), 5))))

## compute scores
(sc <- scores(pred))

## compute mean scores
colMeans(sc)

## plot a (non-randomized) PIT histogram for the predictions
with(pred, pit(x = observed, pdistr = "pnbinom", mu = pred, size = exp(psi)))


## Not run:
######################################################################
# Do one-step-ahead predictions for the models from the Paul & Held
# (2011) paper for the influenza data from Bavaria and Baden-Wuerttemberg
# (this takes some time!)
######################################################################

## see ?hhh4 for a specification of the models

## do 1-step ahead predictions for the last two years

tp <- nrow(fluBYBW)-2*52

val_A0 <- oneStepAhead(res_A0,tp=tp)
val_B0 <- oneStepAhead(res_B0,tp=tp)
val_C0 <- oneStepAhead(res_C0,tp=tp)
```

```
val_A1 <- oneStepAhead(res_A1,tp=tp)
val_B1 <- oneStepAhead(res_B1,tp=tp)
val_C1 <- oneStepAhead(res_C1,tp=tp)

val_A2 <- oneStepAhead(res_A2,tp=tp)
val_B2 <- oneStepAhead(res_B2,tp=tp)
val_C2 <- oneStepAhead(res_C2,tp=tp)

val_D <- oneStepAhead(res_D,tp=tp)


##################################
## compute scores
##################################

#scores
vals <- ls(pattern="val_")
nam <- substring(vals,first=5,last=6)

whichScores <- c("logs", "rps", "ses")
scores_i <- list()
meanScores <- NULL
for(i in seq(along.with=vals)){
  sc <- scores(get(vals[i]), which=whichScores, individual=TRUE)
  scores_i[[i]] <- sc
  meanScores <- rbind(meanScores,colMeans(sc, dims=2))
}

names(scores_i) <- nam
rownames(meanScores) <- nam

##comparison with best model B2

compareWithBest <- function(best, whichModels, nPermut=9999, seed=1234){
  set.seed(seed)
  pVals <- NULL
  for(score in seq_along(whichScores)){
    p <- c()
    for(model in whichModels){
      if(model==best) p <- c(p,NA)
     else p <- c(p,permutationTest(scores_i[[model]][,,score],scores_i[[best]][,,score],
     plot=TRUE,nPermutation=nPermut, verbose=TRUE)$pVal.permut)
    }
    pVals <- cbind(pVals,p)
  }
  return(pVals)
}

pVals_flu <- compareWithBest(best=6, whichModels=1:10,seed=2059710987)
rownames(pVals_flu) <- nam

## End(Not run)
```

hhh4_W *Power-Law and Nonparametric Neighbourhood Weights for* hhh4*-Models*

Description

Set up power-law or nonparametric weights for the neighbourhood component of hhh4-models as proposed by Meyer and Held (2014). Without normalization, power-law weights are $w_{ji} = o_{ji}^{-d}$, where o_{ji} is the order of neighbourhood between regions i and j, see nbOrder, and d is to be estimated. In the nonparametric formulation, maxlag-1 order-specific log-weights are to be estimated (the first-order weight is always fixed to 1 for identifiability).

Usage

```
W_powerlaw(maxlag, normalize = TRUE,
           log = FALSE, initial = if (log) 0 else 1)

W_np(maxlag, to0 = TRUE, normalize = TRUE,
     initial = log(zetaweights(2:maxlag)))
```

Arguments

maxlag	a single integer specifying a limiting order of neighbourhood. If spatial dependence is not to be truncated at some high order, maxlag should be set to the maximum neighbourhood order in the network of regions.
to0	W_np represents order-specific log-weights up to order maxlag. Higher orders are by default (to0=TRUE) assumed to have 0 weight as for W_powerlaw. Alternatively, to0=FALSE requests that the weight at order maxlag should be carried forward to higher orders.
normalize	logical indicating if the weights should be normalized such that the rows of the weight matrix sum to 1 (default).
log	logical indicating if the decay parameter d should be estimated on the log-scale to ensure positivity.
initial	initial value of the parameter vector.

Value

a list which can be passed as a specification of parametric neighbourhood weights in the controlneweights argument of hhh4.

Author(s)

Sebastian Meyer

References

Meyer, S. and Held, L. (2014): Power-law models for infectious disease spread. *The Annals of Applied Statistics*, **8** (3), 1612-1639.
DOI-Link: http://dx.doi.org/10.1214/14-AOAS743

See Also

nbOrder to determine the matrix of neighbourhood orders from a binary adjacency matrix.

siaf.powerlaw, and siaf.step for modelling distance decay as power law or step function in twinstim space-time point process models.

Examples

```
data("measlesWeserEms")

## data contains neighbourhood orders as required for parametric weights
neighbourhood(measlesWeserEms)[1:6,1:6]
max(neighbourhood(measlesWeserEms))  # max order is 5

## fit a power-law decay of spatial interaction
## in a hhh4 model with seasonality and random intercepts in the endemic part
measlesModel <- list(
    ar = list(f = ~ 1),
    ne = list(f = ~ 1, weights = W_powerlaw(maxlag=5, normalize=TRUE, log=FALSE)),
    end = list(f = addSeason2formula(~-1 + ri(), S=1, period=52),
               offset = population(measlesWeserEms)),
    family = "NegBin1")

## fit the model
set.seed(1)  # random intercepts are initialized randomly
measlesFit <- hhh4(measlesWeserEms, measlesModel)
summary(measlesFit)  # "neweights.d" is the decay parameter d

## plot the spatio-temporal weights o_ji^-d / sum_k o_jk^-d
## as a function of neighbourhood order
plot(measlesFit, type="neweights")
## Due to normalization, same distance does not necessarily mean same weight.
## There is no evidence for a power law of spatial interaction in this
## small observation region with only 17 districts.
## A possible simpler model is first-order dependence, i.e., using
## 'weights = neighbourhood(measlesWeserEms) == 1' in the 'ne' component.
```

---

husO104Hosp *Hospitalization date for HUS cases of the STEC outbreak in Germany, 2011*

Description

Data contain the date of hospitalization for 630 hemolytic-uremic syndrome (HUS) cases during the large STEC outbreak in Germany, 2011. Note: Only HUS cases which ultimately had a hospitalization date available/reported are included in the data set. The total number of HUS cases during the outbreak was 855 – see Höhle and an der Heiden (2014) as well as Frank et al. (2011) for details.

For each HUS case the attribute `dHosp` contains the date of hospitalization and the attribute `dReport` contains the date of first arrival of this hospitalization date at the Robert Koch Institute (RKI). As described in Höhle and an der Heiden (2014) the mechanisms of the delay were complicated and should be interpreted with care. For example, the case report could have arrived earlier, but without information about the hospitalization date.

The resulting reporting triangle corresponds to Fig. 1 of the Web appendix of Höhle and an der Heiden (2014). This means that the reports which arrived with a delay longer than 15 days are set to have have arrived after 15 days. Altogether, this gives small discrepancies when compared with the results of the paper. However, as mentioned in the paper, longer delays were not very relevant for the nowcasting.

Usage

```
data(husO104Hosp)
```

Format

A `data.frame` object.

Source

Data were collected during the outbreak as part of the mandatory reporting of notifiable diseases in Germany (Faensen et al., 2006). Here, reports are transmitted from the local health authorities via the state health authorities to the Robert Koch Institute, Berlin. The resulting reporting triangle corresponds to Fig. 1 of the Web appendix of Höhle and an der Heiden (2014).

References

Höhle M and an der Heiden, M (2014). Bayesian Nowcasting during the STEC O104:H4 Outbreak in Germany, 2011, In revision for Biometrics.

Frank C, Werber D, Cramer JP, Askar M, Faber M, an der Heiden M, Bernard H, Fruth A, Prager R, Spode A, Wadl M, Zoufaly A, Jordan S, Kemper MJ, Follin P, Müller L, King LA, Rosner B, Buchholz U, Stark K, Krause G; HUS Investigation Team (2011). Epidemic Profile of Shiga-Toxin Producing Escherichia coli O104:H4 Outbreak in Germany, N Engl J Med. 2011 Nov 10;365(19):1771-80.

Faensen D, Claus H, Benzler J, Ammon A, Pfoch T, Breuer T, Krause G (2014). SurvNet@RKI - a multistate electronic reporting system for communicable diseases, Euro Surveillance, 2006;11(4):100-103.

Examples

```
data("husO104Hosp")
```

imdepi *Occurrence of Invasive Meningococcal Disease in Germany*

Description

imdepi contains data on the spatio-temporal location of 636 cases of invasive meningococcal disease (IMD) caused by the two most common meningococcal finetypes in Germany, 'B:P1.7-2,4:F1-5' (of serogroup B) and 'C:P1.5,2:F3-3' (of serogroup C). imdepifit contains a model fit to the imdepi data.

Usage

```
data(imdepi)
data(imdepifit)
```

Format

imdepi is an object of class "epidataCS" (a list with components events, stgrid, W and qmatrix).

imdepifit is an object of class "twinstim", see summary.twinstim for some simple methods for fitted "twinstim" models.

Details

The imdepi data is a simplified version of what has been analyzed by Meyer et al. (2012). Simplification is with respect to the temporal resolution of the stgrid (see below) to be used in twinstim's endemic model component. In what follows, we describe the elements events, stgrid, W, and qmatrix of imdepi in greater detail.

imdepi$events is a "SpatialPointsDataFrame" object (ETRS89 projection, i.e. EPSG code 3035, with unit 'km') containing 636 events, each with the following entries:

time: Time of the case occurrence measured in number of days since origin. Note that a U(0,1)-distributed random number has been subtracted from each of the original event times (days) to break ties (using untie(imdepi_tied, amount=list(t=1))).

tile: Tile ID in the spatio-temporal grid (stgrid) of endemic covariates, where the event is contained in. This corresponds to one of the 413 districts of Germany.

type: Event type, a factor with levels "B" and "C".

eps.t: Maximum temporal interaction range for the event. Here set to 30 days.

eps.s: Maximum spatial interaction range for the event. Here set to 200 km.

sex: Sex of the case, i.e. a factor with levels "female" and "male". Note: for some cases this information is not available (NA).

agegrp: Factor giving the age group of the case, i.e. 0-2, 3-18 or >=19. Note: for one case this information is not available (NA).

BLOCK, start: Block ID and start time (in days since origin) of the cell in the spatio-temporal endemic covariate grid, which the event belongs to.

popdensity: Population density (per square km) at the location of the event (corresponds to population density of the district where the event is located).

There are further auxiliary columns attached to the events' data the names of which begin with a . (dot): These are created during conversion to the "epidataCS" class and are necessary for fitting the data with twinstim, see the description of the "epidataCS"-class. With coordinates(imdepi$events) one obtains the (x,y) locations of the events.

The district identifier in tile is indexed according to the German official municipality key ("Amtlicher Gemeindeschlüssel"). See http://de.wikipedia.org/wiki/Amtlicher_Gemeindeschl%C3%BCssel for details.

The data component stgrid contains the spatio-temporal grid of endemic covariate information. In addition to the usual bookkeeping variables this includes:

area: Area of the district tile in square kilometers.

popdensity: Population density (inhabitants per square kilometer) computed from DESTATIS (Federal Statistical Office) information (Date: 31.12.2008) on communities level (LAU2) aggregated to district level (NUTS3).

We have actually not included any time-dependent covariates here, we just established this grid with a (reduced -> fast) temporal resolution of *monthly* intervals so that we can model endemic time trends and seasonality (in this discretized time).

The entry W contains the observation window as a "SpatialPolygons" object, in this case the boundaries of Germany. It was obtained as stateD <- rgeos::gUnaryUnion(districtsD), where districtsD represents Germany's districts as at 2009-01-01 (originally obtained from www.geodatenzentrum.de), simplified by the "modified Visvalingam" algorithm (level 6.6%) available at MapShaper.org (v. 0.1.17). The objects districtsD and stateD are contained in system.file("shapes", "districtsl

The entry qmatrix is a 2×2 identity matrix indicating that no transmission between the two finetypes can occur.

Source

IMD case reports: German Reference Centre for Meningococci (NRZM) – hosted by the Department of Hygiene and Microbiology, Julius-Maximilians-Universität Würzburg, Germany. Thanks to Dr. Johannes Elias and Prof. Dr. Ulrich Vogel for providing the data. See http://www.meningococcus.de/ and http://episcangis.hygiene.uni-wuerzburg.de/ for further details.

Shapefile of Germany's districts as at 2009-01-01: Bundesamt für Kartographie und Geodäsie, Frankfurt am Main, Germany, www.geodatenzentrum.de.

References

Meyer, S., Elias, J. and Höhle, M. (2012): A space-time conditional intensity model for invasive meningococcal disease occurrence. *Biometrics*, **68**, 607-616.
DOI-Link: http://dx.doi.org/10.1111/j.1541-0420.2011.01684.x

See Also

the data class "epidataCS", and function twinstim for model fitting.

Examples

```
data("imdepi")

# Basic information
print(imdepi, n=5, digits=2)

# What is an epidataCS-object?
str(imdepi, max.level=4)
names(imdepi$events@data)
# => events data.frame has hidden columns
sapply(imdepi$events@data, class)
# marks and print methods ignore these auxiliary columns

# look at the B type only
imdepiB <- subset(imdepi, type == "B")
#<- subsetting applies to the 'events' component
imdepiB

# select only the last 10 events
tail(imdepi, n=10)   # there is also a corresponding 'head' method

# Access event marks
str(marks(imdepi))

# there is an update-method which assures that the object remains valid
# when changing parameters like eps.s, eps.t or qmatrix
update(imdepi, eps.t = 20)

# Summary
s <- summary(imdepi)
s
str(s)

# Step function of number of infectives
plot(s$counter, xlab = "Time [days]",
     ylab = "Number of infectious individuals",
     main = "Time series of IMD assuming 30 days infectious period")

# distribution of number of potential sources of infection
opar <- par(mfrow=c(1,2), las=1)
for (type in c("B","C")) {
  plot(100*prop.table(table(s$nSources[s$eventTypes==type])),
  xlim=range(s$nSources), xlab = "Number of potential epidemic sources",
  ylab = "Proportion of events [%]")
}
par(opar)

# a histogram of the number of events along time (using the
# plot-method for the epidataCS-class, see ?plot.epidataCS)
opar <- par(mfrow = c(2,1))
plot(imdepi, "time", subset = type == "B", main = "Finetype B")
plot(imdepi, "time", subset = type == "C", main = "Finetype C")
```

```
par(opar)

# Plot the spatial distribution of the events in W
plot(imdepi, "space", points.args = list(col=c("indianred", "darkblue")),
     axes = TRUE, lwd = 2)
title(xlab = "x [km]", ylab = "y [km]")

## Not run:
# or manually (no legends, no account for tied locations)
plot(imdepi$W, lwd=2)
plot(imdepi$events, pch=c(3,4)[imdepi$events$type], cex=0.8,
     col=c("indianred", "darkblue")[imdepi$events$type], add=TRUE)

## End(Not run)

## Not run:
  # Show a dynamic illustration of the spatio-temporal dynamics of the
  # spread during the first year of type B with a step size of 7 days
  animate(imdepiB, interval=c(0,365), time.spacing=7, sleep=0.1)

## End(Not run)
```

`influMen` *Influenza and meningococcal infections in Germany, 2001-2006*

Description

Weekly counts of new influenza and meningococcal infections in Germany 2001-2006.

Usage

```
data(influMen)
```

Format

A `disProg` object containing 312×2 observations starting from week 1 in 2001 to week 52 in 2006.

Source

Robert Koch-Institut: SurvStat: http://www3.rki.de/SurvStat. Queried on 25 July 2007.

Examples

```
data(influMen)
plot(influMen, as.one=FALSE, same.scale=FALSE)
```

`inside.gpc.poly` *Test Whether Points are Inside a* `"gpc.poly"` *Polygon*

Description

Same as, e.g., `inside.owin` from package **spatstat** and `point.in.polygon` from package **sp**, i.e., test whether points lie inside or outside a given polygon. Actually, the method for `"gpc.poly"` documented here internally uses the `point.in.polygon` function.

Usage

```
inside.gpc.poly(x, y = NULL, polyregion, mode.checked = FALSE)
```

Arguments

`x,y`	numeric vectors of coordinates of the points to be tested. The coordinates can be supplied in any form accepted by `xy.coords`.
`polyregion`	an object of class `"gpc.poly"`. It is checked if the points specified through `x` and `y` fall into this polygonal region.
`mode.checked`	passed to `point.in.polygon`.

Details

The nodes and edges of (non-hole) polygons are treated as being inside. Points that fall *strictly* inside holes are treated as being outside of the polygon.

Value

Logical vector whose `i`th entry is `TRUE` if the corresponding point `(x[i],y[i])` is inside `polyregion`.

Author(s)

Sebastian Meyer

Examples

```
if (requireNamespace("rgeos")) {
    poly <- discpoly(c(0.5,0.5), 0.5, npoly=4, class="gpc.poly")
    pts <- cbind(x=runif(50), y=runif(50))
    plot(poly)
    points(pts, col=1+inside.gpc.poly(pts, polyregion=poly))
}
```

intensityplot *Plot Paths of Point Process Intensities*

Description

Generic function for plotting paths of point process intensities. Methods currently defined in package **surveillance** are for classes "twinSIR" and "simEpidata" (temporal), as well as "twinstim" and "simEpidataCS" (spatio-temporal).

Usage

```
intensityplot(x, ...)
```

Arguments

x	An object for which an intensityplot method is defined.
...	Arguments passed to the corresponding method.

See Also

The methods intensityplot.twinSIR and intensityplot.twinstim.

intersectPolyCircle *Intersection of a Polygonal and a Circular Domain*

Description

This is a unifying wrapper around functionality of various packages dealing with spatial data. It computes the intersection of a circular domain and a polygonal domain (whose class defines the specific method).

Usage

```
intersectPolyCircle(object, center, radius, ...)

## S3 method for class 'owin'
intersectPolyCircle(object, center, radius, npoly = 32, ...)
## S3 method for class 'SpatialPolygons'
intersectPolyCircle(object, center, radius, npoly = 32, ...)
## S3 method for class 'gpc.poly'
intersectPolyCircle(object, center, radius, npoly = 32,
                    useGEOS = FALSE, ...)
```

Arguments

`object`	a polygonal domain of one of the supported classes.
`center,radius,npoly`	see `discpoly`.
`useGEOS`	logical indicating if package **rgeos** (`gIntersection`) should be used instead of package **gpclib**. The latter (default) requires explicit acceptance of **gpclib**'s restricted license via `surveillance.options(gpclib=TRUE)`.
`...`	potential further arguments (from the generic).

Value

a polygonal domain of the same class as the input `object`.

Author(s)

Sebastian Meyer

See Also

`discpoly` to generate a polygonal approximation to a disc

Examples

```
data("letterR", package="spatstat")
plot(letterR)
plot(intersectPolyCircle(letterR, c(3,2), 1), add=TRUE, col=2, lwd=3)
```

---

isoWeekYear *Find ISO week and ISO year of a vector of Date objects on Windows*

Description

This function extracts the ISO week and ISO year of a `Date` according to the ISO 8601 specification. Note that this function does nothing else than format.Date(x, "%G") and format.Date(x, "%V") would do on Mac/Unix computers. However, this is not implemented on Windows.

A small internal wrapper for `format.Date` (called `formatDate`) thus directs all calls having one of these format strings to this function, if the `sessionInfo()[[1]]$os` information reveals a Windows system.

Usage

```
isoWeekYear(Y, M=NULL, D=NULL)
```

Arguments

Y	Date object (POSIX) or the year. Can be a vector.
M	month, NULL if Y is a Date object)
D	day, NULL if Y is a Date object)

Details

The code to find the ISO week and year on Windows is by Gustaf Rydevik <gustaf.rydevik_at_gmail.com> posted at http://tolstoy.newcastle.edu.au/R/e10/help/10/05/5588.html

Value

A list with entries ISOYear and ISOWeek containing the corresponding results.

Author(s)

Gustaf Rydevik

Examples

```
dates <- as.Date(c("2002-12-31","2003-01-01","2003-01-06"))
isoWeekYear(dates)
```

---

ks.plot.unif *Plot the ECDF of a uniform sample with Kolmogorov-Smirnov bounds*

Description

This plot function takes a univariate sample that should be tested for a U(0,1) distribution, plots its empirical cumulative distribution function (ecdf), and adds a confidence band by inverting the corresponding Kolmogorov-Smirnov test (ks.test). The uniform distribution is rejected if the ECDF is not completely inside the confidence band.

Usage

```
ks.plot.unif(U, conf.level = 0.95, exact = NULL,
             col.conf = "gray", col.ref = "gray",
             xlab = expression(u[(i)]), ylab = "Cumulative distribution")
```

Arguments

U	numeric vector containing the sample. Missing values are (silently) ignored.
conf.level	confidence level for the K-S-test (defaults to 0.95), can also be a vector of multiple levels.
exact	see ks.test.
col.conf	colour of the confidence lines.
col.ref	colour of the diagonal reference line.
xlab, ylab	axis labels.

Value

NULL (invisibly).

Author(s)

Michael Höhle and Sebastian Meyer. The code contains segments originating from the source of the ks.test function http://svn.r-project.org/R/trunk/src/library/stats/R/ks.test.R, which is Copyright (C) 1995-2012 The R Core Team available under GPL-2 (or later) and C functionality from http://svn.r-project.org/R/trunk/src/library/stats/src/ks.c, which is copyright (C) 1999-2009 the R Core Team and available under GPL-2 (or later). Somewhat hidden in their 'ks.c' file is a statement that part of their code is based on code published in George Marsaglia and Wai Wan Tsang and Jingbo Wang (2003), "Evaluating Kolmogorov's distribution". Journal of Statistical Software, Volume 8, 2003, Issue 18. URL: http://www.jstatsoft.org/v08/i18/.

See Also

ks.test for the Kolmogorov-Smirnov test, as well as checkResidualProcess, which makes use of this plot function.

Examples

```
samp <- runif(99)
ks.plot.unif(samp, conf.level=c(0.95, 0.99), exact=TRUE)
ks.plot.unif(samp, conf.level=c(0.95, 0.99), exact=FALSE)
```

---

layout.labels *Layout Item for Feature Labels in* spplot

Description

Generate an sp.layout item for use in spplot in order to draw labels at the coordinates of the spatial object.

Usage

```
layout.labels(obj, labels = TRUE)
```

Arguments

obj	an object inheriting from a Spatial class.
labels	specification of the labels: • a FALSE or NULL value omits labels (NULL is returned), • labels = TRUE uses row.names(obj), • a character or numeric index for a column of obj@data which contains suitable labels,

- a vector of length `length(obj)` with labels,
- or a list of arguments for `panel.text`, where the optional `labels` component follows the same rules as above.

Value

an `sp.layout` item for `spplot`, which is a list. Its first element is `"panel.text"` and subsequent elements are arguments to that function based on the `labels` specification.

Author(s)

Sebastian Meyer

Examples

```
data("measlesWeserEms")
(li <- layout.labels(measlesWeserEms@map, labels = list(font=2, labels="GEN")))

## example usage in spplot()
spplot(measlesWeserEms@map, zcol = "AREA", sp.layout = list(li),
       col.regions = rev(heat.colors(100)))
```

---

linelist2sts *Convert individual case information based on dates into an aggregated time series*

Description

The function is used to convert an individual line list of cases to an aggregated time series based on date information of the cases.

Usage

```
linelist2sts(linelist,dateCol,aggregate.by="1 week",dRange=NULL,
            startYearFormat=switch(aggregate.by,
"1 day"="%V","7 day"="%V","1 week"="%V","1 month"="%Y","3 month"="%Y"),
           startEpochFormat=switch(aggregate.by,
"1 day"="%j","7 day"="%V","1 week"="%V","1 month"="%m","3 month"="%Q")
)
```

Arguments

`linelist`	A `data.frame` containing the line list of cases.
`dateCol`	A character string stating the column name in `linelist` which contain the case data which are to be temporally aggregated.
`aggregate.by`	Temporal aggregation level given as a string, see the `by` variable of the `seq.Date` function for further details.

dRange — A vector containing the minimum and maximum data to use. If not specified these dates are extracted automatically by taking `range(D[,dateCol])`.

startYearFormat — Strptime compatible format string to use for determining how the date string is generated. Usually the provided options are sufficient.

startEpochFormat — Strptime compatible format string to use for determining how the date string is generated. Usually the provided options are sufficient.

Details

In case aggregation occurs by week the date range is automatically extended such that the starting and ending dates are mondays. This might not be an appropriate way in all situations this function is to be used.

Value

The function returns an object of class "sts". The freq slot might not be appropriate.

Note

This implementation is still experimental, changes might occur in the future.

Author(s)

Michael Höhle

See Also

See also seq.Date.

Examples

```
#Load simulated outbreak data.
url <- paste("http://www.stat.uni-muenchen.de/~hoehle/",
             "teaching/moid2011/tutorials/cast-backnow/outbreak.txt",sep="")
D <- try(read.table(url,header=TRUE,colClasses=c("integer",rep("Date",3))))

if (!inherits(D, "try-error")) {
    #Convert line list to an sts object
    sts <- linelist2sts(D, dateCol="dOnset", aggregate.by="1 day")

    #Plot the result
    plot(sts,xaxis.tickFreq=list("%d"=atChange,"%m"=atChange),
             xaxis.labelFreq=list("%d"=at2ndChange),
             xaxis.labelFormat="%d %b",legend.opts=NULL,
             xlab="",las=2,cex.axis=0.8)
}
```

LRCUSUM.runlength *Run length computation of a CUSUM detector*

Description

Compute run length for a count data or categorical CUSUM. The computations are based on a Markov representation of the likelihood ratio based CUSUM.

Usage

```
LRCUSUM.runlength(mu,mu0,mu1,h,dfun, n, g=5,outcomeFun=NULL,...)
```

Arguments

mu	$k-1 \times T$ matrix with true proportions, i.e. equal to mu0 or mu1 if one wants to compute e.g. ARL_0 or ARL_1.
mu0	$k-1 \times T$ matrix with in-control proportions
mu1	$k-1 \times T$ matrix with out-of-control proportion
h	The threshold h which is used for the CUSUM.
dfun	The probability mass function or density used to compute the likelihood ratios of the CUSUM. In a negative binomial CUSUM this is dnbinom, in a binomial CUSUM dbinom and in a multinomial CUSUM dmultinom.
n	Vector of length T containing the total number of experiments for each time point.
g	The number of levels to cut the state space into when performing the Markov chain approximation. Sometimes also denoted M. Note that the quality of the approximation depends very much on g. If T greater than, say, 50 its necessary to increase the value of g.
outcomeFun	A hook function to compute all possible outcome states to compute the likelihood ratio for. If NULL then the default function outcomeFunStandard(k,n) is used. This function uses the Cartesian product of 0:n for k components.
...	Additional arguments to send to dfun.

Details

Brook and Evans (1972) formulated an approximate approach based on Markov chains to determine the PMF of the run length of a time-constant CUSUM detector. They describe the dynamics of the CUSUM statistic by a Markov chain with a discretized state space of size $g + 2$. This is adopted to the time varying case in Höhle (2010) and implemented in R using the ... notation such that it works for a very large class of distributions.

Value

A list with five components

P	An array of $g+2 \times g+2$ transition matrices of the approximation Markov chain.
pmf	Probability mass function (up to length T) of the run length variable.
cdf	Cumulative density function (up to length T) of the run length variable.
arl	If the model is time homogenous (i.e. if $T == 1$) then the ARL is computed based on the stationary distribution of the Markov chain. See the eqns in the reference for details. Note: If the model is not time homogeneous then the function returns NA and the ARL has to be approximated manually from the output. One could use `sum(1:length(pmf) * pmf)`, which is an approximation because of using a finite support for a sum which should be from 1 to infinity.

Author(s)

M. Höhle

References

Höhle, M. (2010), Changepoint detection in categorical time series, Book chapter in T. Kneib and G. Tutz (Eds.), Statistical Modelling and Regression Structures - Festschrift in Honour of Ludwig Fahrmeir, Springer, pp. 377-397. Preprint available as http://www.math.su.se/~hoehle/pubs/hoehle2010-preprint.pdf

Höhle, M. and Mazick, A. (2009), Aberration detection in R illustrated by Danish mortality monitoring, Book chapter to appear in T. Kass-Hout and X. Zhang (Eds.) Biosurveillance: A Health Protection Priority, CRCPress. Preprint available as http://www.math.su.se/~hoehle/pubs/hoehle_mazick2009-preprint.pdf

Brook, D. and Evans, D. A. (1972), An approach to the probability distribution of Cusum run length, Biometrika, 59:3, pp. 539–549.

See Also

categoricalCUSUM

Examples

```
######################################################
#Run length of a time constant negative binomial CUSUM
######################################################

#In-control and out of control parameters
mu0 <- 10
alpha <- 1/2
kappa <- 2

#Density for comparison in the negative binomial distribution
dY <- function(y,mu,log=FALSE, alpha, ...) {
  dnbinom(y, mu=mu, size=1/alpha, log=log)
}
```

```
#In this case "n" is the maximum value to investigate the LLR for
#It is assumed that beyond n the LLR is too unlikely to be worth
#computing.
LRCUSUM.runlength( mu=t(mu0), mu0=t(mu0), mu1=kappa*t(mu0), h=5,
  dfun = dY, n=rep(100,length(mu0)), alpha=alpha)

h.grid <- seq(3,6,by=0.3)
arls <- sapply(h.grid, function(h) {
  LRCUSUM.runlength( mu=t(mu0), mu0=t(mu0), mu1=kappa*t(mu0), h=h,
  dfun = dY, n=rep(100,length(mu0)), alpha=alpha,g=20)$arl
})
plot(h.grid, arls,type="l",xlab="threshold h",ylab=expression(ARL[0]))


if (surveillance.options("allExamples"))
{
  ######################################################
  #Run length of a time varying negative binomial CUSUM
  ######################################################

  mu0 <- matrix(5*sin(2*pi/52 * 1:200) + 10,ncol=1)

  rl <- LRCUSUM.runlength( mu=t(mu0), mu0=t(mu0), mu1=kappa*t(mu0), h=2,
    dfun = dY, n=rep(100,length(mu0)), alpha=alpha,g=20)

  plot(1:length(mu0),rl$pmf,type="l",xlab="t",ylab="PMF")
  plot(1:length(mu0),rl$cdf,type="l",xlab="t",ylab="CDF")
}


#########################################################
# Further examples contain the binomial, beta-binomial
# and multinomial CUSUMs. Hopefully, these will be added
# in the future.
#########################################################

#dfun function for the multinomial distribution (Note: Only k-1 categories are specified).
dmult <- function(y, size,mu, log = FALSE) {
    return(dmultinom(c(y,size-sum(y)), size = size, prob=c(mu,1-sum(mu)), log = log))
}

#Example for the time-constant multinomial distribution
#with size 100 and in-control and out-of-control parameters as below.
n <- 100
pi0 <- as.matrix(c(0.5,0.3,0.2))
pi1 <- as.matrix(c(0.38,0.46,0.16))

#ARL_0
LRCUSUM.runlength(mu=pi0[1:2,,drop=FALSE],mu0=pi0[1:2,,drop=FALSE],mu1=pi1[1:2,,drop=FALSE],
                  h=5,dfun=dmult, n=n, g=15)$arl
#ARL_1
LRCUSUM.runlength(mu=pi1[1:2,,drop=FALSE],mu0=pi0[1:2,,drop=FALSE],mu1=pi1[1:2,,drop=FALSE],
```

```
h=5,dfun=dmult, n=n, g=15)$arl
```

`m1` *RKI SurvStat Data*

Description

14 datasets for different diseases beginning in 2001 to the 3rd Quarter of 2004 including their defined outbreaks.

- `m1` 'Masern' in the 'Landkreis Nordfriesland' (Germany, Schleswig-Holstein)
- `m2` 'Masern' in the 'Stadt- und Landkreis Coburg' (Germany, Bayern)
- `m3` 'Masern' in the 'Kreis Leer' (Germany, Niedersachsen)
- `m4` 'Masern' in the 'Stadt- und Landkreis Aachen' (Germany, Nordrhein-Westfalen)
- `m5` 'Masern' in the 'Stadt Verden' (Germany, Niedersachsen)
- `q1_nrwh` 'Q-Fieber' in the 'Hochsauerlandkreis' (Germany, Westfalen) and in the 'Landkreis Waldeck-Frankenberg' (Germany, Hessen)
- `q2` 'Q-Fieber' in 'München' (Germany, Bayern)
- `s1` 'Salmonella Oranienburg' in Germany
- `s2` 'Salmonella Agona' in 12 'Bundesländern' of Germany
- `s3` 'Salmonella Anatum' in Germany
- `k1` 'Kryptosporidiose' in Germany, 'Baden-Württemberg'
- `n1` 'Norovirus' in 'Stadtkreis Berlin Mitte' (Germany, Berlin)
- `n2` 'Norovirus' in 'Torgau-Oschatz' (Germany, Sachsen)
- `h1_nrwrp` 'Hepatitis A' in 'Oberbergischer Kreis, Olpe, Rhein-Sieg-kreis' (Germany, Nordrhein-Westfalen) and 'Siegenwittgenstein Altenkirchen' (Germany, Rheinland-Pfalz)

Usage

```
data(m1)
```

Format

`disProg` objects each containing 209 observations (weekly on 52 weeks)

observed Number of counts in the corresponding week

state Boolean whether there was an outbreak.

Source

Robert Koch-Institut: SurvStat: http://www3.rki.de/SurvStat; m1 and m3 were queried on 10 November 2004. The rest during September 2004.

See Also

readData

Examples

```
data(k1)
survResObj <- algo.rki1(k1, control=list(range=27:192))
plot(survResObj, "RKI 1", "k1", firstweek=27, startyear=2002)
```

---

magic.dim *Returns a suitable k1 x k2 for plotting the disProgObj*

Description

For a given number k magic.dim provides a vector containing two elements, the number of rows (k1) and columns (k2), respectively, which can be used to set the dimension of a single graphic device so that k1*k2 plots can be drawn by row (or by column) on the device.

Usage

```
magic.dim(k)
```

Arguments

k	an integer

Value

vector with two elements

See Also

primeFactors and bestCombination which are internally used to complete the task.

n2mfrow is a similar function from package **grDevices**.

makePlot *Plot Generation*

Description

Just a test method.

Usage

```
makePlot(outputpath, data = "k1", method = "rki1",
                             name, disease, range = 157:339)
```

Arguments

outputpath	path for the storage
data	abbreviation of the disease-file
method	method to be called
name	name of the method
disease	disease name
range	range to plot

Details

makePlot reads the data given in data using the function readData, and the data are corrected to 52 weeks, enlarged using enlargeData and sendt to the surveillance system given in method. The system result is plotted and stored in outputpath.

Author(s)

M. Höhle, A. Riebler, C. Lang

See Also

readData, correct53to52, enlargeData, algo.call, plot.survRes

Examples

```
makePlot("./", "k1", "rki2", "RKI 2", "Kryptosporidiose")
```

marks *Import from package* **spatstat**

Description

The generic function marks is imported from package **spatstat**. See spatstat::marks for **spatstat**'s own methods, and marks.epidataCS for the "epidataCS"-specific method.

`measles.weser` *Measles in the Weser-Ems region of Lower Saxony, Germany, 2001-2002*

Description

Weekly counts of new measles cases for the 17 administrative districts (NUTS-3 level) of the "Weser-Ems" region of Lower Saxony, Germany, during 2001 and 2002, as reported to the Robert Koch institute according to the Infection Protection Act ("Infektionsschutzgesetz", IFSG).
`data("measlesWeserEms")` is a corrected version of `data("measles.weser")` (see Format section below).

Usage

```
data("measles.weser")
data("measlesWeserEms")
```

Format

`data("measles.weser")` is an object of the old `"disProg"` class, whereas `data("measlesWeserEms")` is of the new class `"sts"`.

Furthermore, the following updates have been applied for `data("measlesWeserEms")`:

- it includes the two districts "SK Delmenhorst" (03401) and "SK Wilhemshaven" (03405) with zero counts, which are ignored in `data("measles.weser")`.
- it corrects the time lag error for year 2002 caused by a redundant pseudo-week "0" with 0 counts only (the row `measles.weser$observed[53,]` is nonsense).
- it has one more case attributed to "LK Oldenburg" (03458) during 2001/W17, i.e., 2 cases instead of 1. This reflects the official data as of "Jahrbuch 2005", whereas `data("measles.weser")` is as of "Jahrbuch 2004".
- it contains a map of the region (as a `"SpatialPolygonsDataFrame"`) with the following variables:

  `GEN` district label.

  `AREA` district area in m^2.

  `POPULATION` number of inhabitants (as of 31/12/2003).

  `vaccdoc.2004` proportion with a vaccination card among screened abecedarians (2004).

  `vacc1.2004` proportion with at least one vaccination against measles among abecedarians presenting a vaccination card (2004).

  `vacc2.2004` proportion of doubly vaccinated abecedarians among the ones presenting their vaccination card at school entry in the year 2004.
- it uses the correct format for the official district keys, i.e., 5 digits (initial 0).
- its attached neighbourhood matrix is more general: a distance matrix (neighbourhood orders) instead of just an adjacency indicator matrix (special case `nbOrder == 1`).
- population fractions represent data as of 31/12/2003 (NLS, 2004, document "A I 2 - hj 2 / 2003"). There are only minor differences to the ones used for `data("measles.weser")`.

Source

Measles counts were obtained from the public SurvStat database of the Robert Koch institute: http://www3.rki.de/SurvStat.

A shapefile of Germany's districts as of 01/01/2009 was obtained from the Service Center (www.geodatenzentrum.de) of the German Federal Agency for Cartography and Geodesy (www.bkg.bund.de). The map of the 17 districts of the "Weser-Ems" region (`measlesWeserEms@map`) is a simplified subset of this shapefile using a 30% reduction via the Douglas-Peucker reduction method as implemented at MapShaper.org.

Population numbers are obtained from the Federal Statistical Office of Lower Saxony (NLS): http://www.lskn.niedersachsen.de/portal/live.php?navigation_id=25688&article_id=87679&_psmand=40

Vaccination coverage was obtained from the public health department of Lower Saxony: Niedersächsisches Landesgesundheitsamt (2005): Impfreport – Durchimpfung von Kindern im Einschulungsalter in Niedersachsen im Erhebungsjahrgang 2004. Online available from http://www.nlga.niedersachsen.de/portal/live.php?navigation_id=27093&article_id=19385&_psmand=20, also as an interactive version.

Examples

```
data("measles.weser")
measles.weser
plot(measles.weser, as.one=FALSE)

data("measlesWeserEms")
measlesWeserEms
plot(measlesWeserEms)
```

---

measlesDE *Measles in the 16 states of Germany*

Description

Weekly number of measles cases in the 16 states (Bundeslaender) of Germany for years 2005 to 2007.

Usage

```
data(measlesDE)
```

Format

An `sts` object containing 156×16 observations starting from week 1 in 2005.

The `population` slot contains the population fractions of each state at 31.12.2006, obtained from the Federal Statistical Office of Germany.

Source

Robert Koch-Institut: SurvStat: http://www3.rki.de/SurvStat; Queried on 14 October 2009.

References

Herzog, S.A., Paul, M. and Held, L. (2011) Heterogeneity in vaccination coverage explains the size and occurrence of measles epidemics in German surveillance data. Epidemiology and Infection, **139**, 505–515.

See Also

MMRcoverageDE

Examples

```
data(measlesDE)
plot(measlesDE)
```

---

meningo.age *Meningococcal infections in France 1985-1995*

Description

Monthly counts of meningococcal infections in France 1985-1995. Here, the data is split into 4 age groups (<1, 1-5, 5-20, >20).

Usage

```
data(meningo.age)
```

Format

An multivariate object of class disProg with 156 observations in each one of 4 age groups.

week Number of month

observed Matrix with number of counts in the corresponding month and age group

state Boolean whether there was an outbreak – dummy not implemented

neighbourhood Neighbourhood matrix, all age groups are adjacent

populationFrac Population fractions

Source

??

Examples

```
data(meningo.age)
plot(meningo.age, title="Meningococcal infections in France 1985-95")
plot(meningo.age, as.one=FALSE)
```

MMRcoverageDE *MMR coverage levels in the 16 states of Germany*

Description

Coverage levels at school entry for the first and second dose of the combined measles-mumps-rubella (MMR) vaccine in 2006, estimated from children presenting vaccination documents at school entry examinations.

Usage

```
data(MMRcoverageDE)
```

Format

A `data.frame` containing 19 rows and 5 columns with variables

state Names of states: first each of the 16 states (Bundeslaender, BL) is listed, followed by the Total of Germany, as well as the total of West (alte BL) and East Germany (neue BL).

nOfexaminedChildren Number of children examined.

withVaccDocument Percentage of children who presented vaccination documents.

MMR1 Percentage of children with vaccination documents, who received at least 1 dose of MMR vaccine.

MMR2 Percentage of children with vaccination documents, who received at least 2 doses of MMR vaccine.

Coverage levels were derived from vaccination documents presented at medical examinations, which are conducted by local health authorities at school entry each year. Records include information about the receipt of 1st and 2nd doses of MMR, but no information about dates. Note that information from children who did not present a vaccination document on the day of the medical examination, is not included in the estimated coverage.

Source

Robert Koch-Institut (2008) Zu den Impfquoten bei den Schuleingangsuntersuchungen in Deutschland 2006. Epidemiologisches Bulletin, **7**, 55-57

References

Herzog, S.A., Paul, M. and Held, L. (2011) Heterogeneity in vaccination coverage explains the size and occurrence of measles epidemics in German surveillance data. Epidemiology and Infection, **139**, 505–515.

See Also

measlesDE

Examples

```
data(MMRcoverageDE)
```

---

momo *Danish 1994-2008 all cause mortality data for six age groups*

Description

Weekly number of all cause mortality from 1994-2008 in each of the six age groups <1, 1-4, 5-14, 15-44, 45-64, 65-74, 75-84 and 85 years.

Usage

```
data(momo)
```

Details

The object of class `sts` contains the number of all cause mortality from 1994-2008 in Denmark for each of the six age groups <1, 1-4, 5-14, 15-44, 45-64, 65-74, 75-84 and 85 years. A special feature of such EuroMOMO data is that weeks are handled as defined by the ISO 8601 standard, which can be handled by the `sts` class.

The `population` slot of the `momo` object contains the population size in each of the six age groups. These are yearly data obtained from the StatBank Denmark.

The aim of the EuroMOMO project is to develop and strengthen real-time monitoring of mortality across Europe; this will enhance the management of serious public health risks such as pandemic influenza, heat waves and cold snaps. For further details see the homepage of the EuroMOMO project.

Source

Department of Epidemiology, Statens Serum Institute, Copenhagen, Denmark StatBank Denmark, Statistics Denmark, http://www.statistikbanken.dk/

References

Höhle, M. and A. Mazick, A. (2009) Aberration detection in R illustrated by Danish mortality monitoring, Book chapter to appear in T. Kass-Hout and X. Zhang (Eds.) Biosurveillance: A Health Protection Priority, CRC Press.

EuroMOMO project page, http://www.euromomo.eu/, Last accessed: 13 Oct 2010.

Examples

```
data("momo")
plot(momo,legend.opts=NULL)
```

multiplicity *Import from package* **spatstat**

Description

The generic function `multiplicity` is imported from package **spatstat**. See `spatstat::multiplicity` for **spatstat**'s own methods, and `multiplicity.Spatial` for the added method for `Spatial` objects.

multiplicity.Spatial *Count Number of Instances of Points*

Description

The generic function `multiplicity` defined in **spatstat** is intended to count the number of duplicates of each element of an object. **spatstat** already offers methods for point patterns, matrices and data frames, and here we add a method for `Spatial` objects from the **sp** package. It is a wrapper for the default method, which effectively computes the distance matrix of the points, and then just counts the number of zeroes in each row.

Usage

```
## S3 method for class 'Spatial'
multiplicity(x)
```

Arguments

`x`	a `"Spatial"` object (we only need a `coordinates`-method), e.g. of class `"SpatialPoints"`.

Value

an integer vector containing the number of instances of each point of the object.

See Also

`multiplicity` in package **spatstat**. See the Examples of the `hagelloch` data for a specific use of `multiplicity`.

Examples

```
foo <- SpatialPoints(matrix(c(1,2,
                              2,3,
                              1,2,
                              4,5), 4, 2, byrow=TRUE))
multiplicity(foo)

# the following function determines the multiplicities in a matrix
# or data frame and returns unique rows with appended multiplicity
countunique <- function(x) unique(cbind(x, count=multiplicity(x)))
countunique(coordinates(foo))
```

`nbOrder` *Determine Neighbourhood Order Matrix from Binary Adjacency Matrix*

Description

Given a square binary adjacency matrix, the function `nbOrder` determines the integer matrix of neighbourhood orders (shortest-path distance) using the function `nblag` from the **spdep** package.

Usage

```
nbOrder(neighbourhood, maxlag = 1)
```

Arguments

`neighbourhood`	a square, numeric or logical, and usually symmetric matrix with finite entries (and usually zeros on the diagonal) which indicates vertex adjacencies, i.e., first-order neighbourhood (interpreted as `neighbourhood == 1`, *not* `>0`).
`maxlag`	positive scalar integer specifying an upper bound for the neighbourhood order. The default (1) just returns the input neighbourhood matrix (converted to binary integer mode). `maxlag` is automatically trimmed to one less than the number of regions (there cannot be higher orders) and then converted to integer, thus, `maxlag = Inf` also works.

Value

An integer matrix of neighbourhood orders, i.e., the shortest-path distance matrix of the vertices. The `dimnames` of the input `neighbourhood` matrix are preserved.

Note

By the end, the function issues a `message` informing about the range of maximum neighbourhood order by region.

Author(s)

Sebastian Meyer

See Also

nblag from the **spdep** package, on which this wrapper depends.

Examples

```
## generate adjacency matrix
set.seed(1)
n <- 6
adjmat <- matrix(0, n, n)
adjmat[lower.tri(adjmat)] <- sample(0:1, n*(n-1)/2, replace=TRUE)
adjmat <- adjmat + t(adjmat)
adjmat

## determine neighbourhood order matrix
if (requireNamespace("spdep")) {
    nbmat <- nbOrder(adjmat, maxlag=Inf)
    nbmat
}
```

---

nowcast *Adjust a univariate time series of counts for observed but-not-yet-reported events*

Description

Nowcasting can help to obtain up-to-date information on trends during a situation where reports about events arrive with delay. For example in any type of public health reporting, reports about important indicators (such as occurrence of cases) are prone to be delayed due to for example manual quality checking and reporting system hierarchies. Altogether, delays are subject to a delay distribution, which may or may not vary over time.

Usage

```
nowcast(now,when,data,dEventCol="dHospital",dReportCol="dReport",
        method=c("bayes.notrunc","bayes.notrunc.bnb","lawless","bayes.trunc",
                 "unif","bayes.trunc.ddcp"),
        aggregate.by="1 day",
        D=15, m=NULL,
        control=list(
            dRange=NULL,alpha=0.05,nSamples=1e3,
            N.tInf.prior=c("poisgamma","pois","unif"),
            N.tInf.max=300, gd.prior.kappa=0.1,
            ddcp=list(ddChangepoint=NULL,
                logLambda=c("iidLogGa","tps","rw1","rw2"),
```

```
    tau.gamma=1,eta.mu=NULL, eta.prec=NULL,
    mcmc=c(burnin=2500,sample=10000,thin=1)),
score=FALSE,predPMF=FALSE))
```

Arguments

now	an object of class Date denoting the day at which to do the nowcast. This corresponds to T in the notation of Höhle and an der Heiden (2014).
when	a vector of Date objects denoting the day(s) for which the projections are to be done. One needs to ensure that each element in when is smaller or equal to now.
data	A data frame with one row per case – for each case on needs information on the day of the event (e.g. hospitalization) and the day of report of this event.
dEventCol	The name of the column in data which contains the date of the event, e.g. hospitalization. Default: "dHospital".
dReportCol	Name of the column in data containing the date at which the report arrives at the respective register. Default: "dReport".
method	A vector of strings denoting the different methods to use to nowcast. Note that results of the first name in this list are officially returned by the function. However, it is possible to specify several methods here, e.g. in order to compare score evaluations. Details of the methods are described in Höhle and an der Heiden (2014). **code"unif"** **code"bayes.notrunc"** A Bayesian procedure ignoring truncation. **code"bayes.notrunc.bnb"** **code"lawless"** A discretized version of the Gaussian predictive distribution suggested in Lawless (1994). **code"bayes.trunc"** Bayesian method based on the generalized Dirichlet distribution, which is the conjugate prior-posterior for the delay distribution PMF under right-truncated sampling as shown in HadH (2014). **code"bayes.trunc.ddcp"**
aggregate.by	Time scale used for the temporal aggregation of the records in the data data. See linelist2sts and seq.Date for further information.
D	Maximum possible or maximum relevant delay (unit: aggregate.by). Default: 15.
m	Moving window for the estimation of the delay distribution. Default: NULL, i.e. take all values at all times.
control	A list with named arguments controlling the functionality of the nowcasting. **dRange** Default: NULL. In this case the dEventCol column is used to extract the first and last available in data. **alpha** Equal tailed (1-α)*100% prediction intervals are calculated. Default: 0.05. **nSamples** Number of PMF samples in the bayes.* procedures. Note: Entire vectors containing the PMF on the grid from 0 to N.tInf.max are drawn and which are then combined. The argument does not apply to the bayes.trunc.ddcp method.

N.tInf.prior Prior distribution of $N(t, \infty)$. Applies only to the `bayes.*` except `bayes.bayes.ddcp` methods. See example on how to control the distribution parameters.

N.tInf.max Limit of the support of $N(t, \infty)$. The value needs to be high enough such that at this limit only little of the predictive distribution is right-truncated. Default: 300.

gd.prior.kappa Concentration parameter for the Dirichlet prior for the delay distribution on $0, ..., D$. Default: 0.1. Note: The procedure is quite sensitive to this parameters in case only few cases are available.

ddcp A list specifying the change point model for the delay distribution. This method should only be used if detailed information about changes in the delay distribution are available as, e.g., in the case of the STEC O104:H4 outbreak. The components are as follows:

`ddChangepoint` Vector of Date objects corresponding to the changepoints

`logLambda` Prior on the spline. One of `c("iidLogGa","tps","rw1","rw2")`.

`tau.gamma`

`eta.mu`

`eta.prec`

`mcmc` A names vector of length 3 containing burn-in, number of samples and thinning for the three MCMC chains which are ran. The values are passed on to `runjags`. Default: `c(burnin=2500,sample=10000,thin=1)`.

score Compute scoring rules. Default: `FALSE`. The computed scores are found in the `SR` slot of the result.

predPMF Boolean whether to teturn the probability mass functions of the individual forecasts (Default: `FALSE`). The result can be found in the `control` slot of the return object.

Details

The methodological details of the nowcasting procedures are described in Höhle M and an der Heiden M (2014).

Value

`nowcast` returns an object of `"stsNC"`. The `upperbound` slot contains the median of the method specified at the first position the argument `method`. The slot `pi` (for prediction interval) contains the equal tailed (1-α)*100% prediction intervals, which are calculated based on the predictive distributions in slot `predPMF`. Furthermore, slot `truth` contains an `sts` object containing the true number of cases (if possible to compute it based on the data in `data`. Finally, slot `SR` contains the results for the proper scoring rules (requires truth to be calculable).

Note

Note: The `bayes.trunc.ddcp` uses the JAGS software together with the R package `runjags` to handle the parallelization of the MCMC using the runjags `rjparallel` method. You need to manually install JAGS on your computer for the package to work – see http://mcmc-jags.sourceforge.net/ and the documentation of `runjags` for details.

Note: The function is still under development and might change in the future. Unfortunately, little emphasis has so far been put on making the function easy to understand and use.

Author(s)

Michael Höhle

References

Höhle M and an der Heiden M (2014), Bayesian Nowcasting during the STEC O104:H4 Outbreak in Germany, 2011, Biometrics, http://dx.doi.org/10.1111/biom.12194.

Examples

```
data("husO104Hosp")

#Extract the reporting triangle at a specific day
t.repTriangle <- as.Date("2011-07-04")

#Use 'void' nowcasting procedure (we just want the reporting triangle)
nc <- nowcast(now=t.repTriangle,when=t.repTriangle,
              dEventCol="dHosp",dReportCol="dReport",data=husO104Hosp,
              D=15,method="unif")

#Show reporting triangle
reportingTriangle(nc)

#Perform Bayesian nowcasting assuming the delay distribution is stable over time
nc.control <- list(N.tInf.prior=structure("poisgamma",
                                mean.lambda=50,var.lambda=3000),
                                nSamples=1e2)

t.repTriangle <- as.Date("2011-06-10")
when <- seq(t.repTriangle-3,length.out=10,by="-1 day")
nc <- nowcast(now=t.repTriangle,when=when,
              dEventCol="dHosp",dReportCol="dReport",data=husO104Hosp,
              D=15,method="bayes.trunc",control=nc.control)

#Show time series and posterior median forecast/nowcast
plot(nc,xaxis.tickFreq=list("%d"=atChange,"%m"=atChange),
     xaxis.labelFreq=list("%d"=at2ndChange),xaxis.labelFormat="%d-%b",
     xlab="Time (days)",lty=c(1,1,1,1),lwd=c(1,1,2))
```

`pairedbinCUSUM` *Paired binary CUSUM and its run-length computation*

Description

CUSUM for paired binary data as described in Steiner et al. (1999).

Usage

```
pairedbinCUSUM(stsObj, control = list(range=NULL,theta0,theta1,
                                      h1,h2,h11,h22))
pairedbinCUSUM.runlength(p,w1,w2,h1,h2,h11,h22, sparse=FALSE)
```

Arguments

stsObj	Object of class sts containing the paired responses for each of the, say n, patients. The observed slot of stsObj is thus a $n \times 2$ matrix.
control	Control object as a list containing several parameters. • rangeVector of indices in the observed slot to monitor. • theta0In-control parameters of the paired binary CUSUM. • theta1Out-of-control parameters of the paired binary CUSUM. • h1Primary control limit (=threshold) of 1st CUSUM. • h2Primary control limit (=threshold) of 2nd CUSUM. • h11Secondary limit for 1st CUSUM. • h22Secondary limit for 2nd CUSUM.
p	Vector giving the probability of the four different possibile states, i.e. c((death=0,near-miss=0),(death=1,near-miss=0), (death=0,near-miss=1),(death=1,near-miss=1)).
w1	The parameters w1 and w2 are the sample weights vectors for the two CUSUMs, see eqn. (2) in the paper. We have that w1 is equal to deaths
w2	As for w1
h1	decision barrier for 1st individual cusums
h2	decision barrier for 2nd cusums
h11	together with h22 this makes up the joing decision barriers
h22	together with h11 this makes up the joing decision barriers
sparse	Boolean indicating whether to use sparse matrix computations from the Matrix library (usually much faster!). Default: FALSE.

Details

For details about the method see the Steiner et al. (1999) reference listed below. Basically, two individual CUSUMs are run based on a logistic regression model. The combined CUSUM not only signals if one of its two individual CUSUMs signals, but also if the two CUSUMs simultaneously cross the secondary limits.

Value

An sts object with observed, alarm, etc. slots trimmed to the control$range indices.

Author(s)

S. Steiner and M. Höhle

References

Steiner, S. H., Cook, R. J., and Farewell, V. T. (1999), Monitoring paired binary surgical outcomes using cumulative sum charts, Statistics in Medicine, 18, pp. 69–86.

See Also

categoricalCUSUM

Examples

```
#Set in-control and out-of-control parameters as in paper
theta0 <- c(-2.3,-4.5,2.5)
theta1 <- c(-1.7,-2.9,2.5)

#Small helper function to compute the paired-binary likelihood
#of the length two vector yz when the true parameters are theta
dPBin <- function(yz,theta) {
    exp(dbinom(yz[1],size=1,prob=plogis(theta[1]),log=TRUE) +
    dbinom(yz[2],size=1,prob=plogis(theta[2]+theta[3]*yz[1]),log=TRUE))
}

#Likelihood ratio for all four possible configurations
p <- c(dPBin(c(0,0), theta=theta0), dPBin(c(0,1), theta=theta0),
       dPBin(c(1,0), theta=theta0), dPBin(c(1,1), theta=theta0))

#Compute ARL using non-sparse matrix operations
## Not run:
pairedbinCUSUM.runlength(p,w1=c(-1,37,-9,29),w2=c(-1,7),h1=70,h2=32,h11=38,h22=17)

## End(Not run)

#Sparse computations don't work on all machines (e.g. the next line
#might lead to an error. If it works this call can be considerably (!) faster
#than the non-sparse call.
## Not run:
pairedbinCUSUM.runlength(p,w1=c(-1,37,-9,29),w2=c(-1,7),h1=70,h2=32,
                         h11=38,h22=17,sparse=TRUE)

## End(Not run)

#Use paired binary CUSUM on the De Leval et al. (1994) arterial switch
#operation data on 104 newborn babies
data("deleval")

#Switch between death and near misses
observed(deleval) <- observed(deleval)[,c(2,1)]

#Run paired-binary CUSUM without generating alarms.
pb.surv <- pairedbinCUSUM(deleval,control=list(theta0=theta0,
             theta1=theta1,h1=Inf,h2=Inf,h11=Inf,h22=Inf))

plot(pb.surv, xaxis.labelFormat=NULL)
```

```
#########################################################################
#Scale the plots so they become comparable to the plots in Steiner et
#al. (1999). To this end a small helper function is defined.
#########################################################################

#########################################################################
#Log LR for conditional specification of the paired model
#########################################################################
LLR.pairedbin <- function(yz,theta0, theta1) {
    #In control
    alphay0 <- theta0[1] ; alphaz0 <- theta0[2] ; beta0 <- theta0[3]
    #Out of control
    alphay1 <- theta1[1] ; alphaz1 <- theta1[2] ; beta1 <- theta1[3]
    #Likelihood ratios
    llry <- (alphay1-alphay0)*yz[1]+log(1+exp(alphay0))-log(1+exp(alphay1))
    llrz <- (alphaz1-alphaz0)*yz[2]+log(1+exp(alphaz0+beta0*yz[1]))-
                                    log(1+exp(alphaz1+beta1*yz[1]))
    return(c(llry=llry,llrz=llrz))
}


val <- expand.grid(0:1,0:1)
table <- t(apply(val,1, LLR.pairedbin, theta0=theta0, theta1=theta1))
w1 <- min(abs(table[,1]))
w2 <- min(abs(table[,2]))
S <- upperbound(pb.surv) / cbind(rep(w1,nrow(observed(pb.surv))),w2)

#Show results
par(mfcol=c(2,1))
plot(1:nrow(deleval),S[,1],type="l",main="Near Miss",xlab="Patient No.",
     ylab="CUSUM Statistic")
lines(c(0,1e99), c(32,32),lty=2,col=2)
lines(c(0,1e99), c(17,17),lty=2,col=3)

plot(1:nrow(deleval),S[,2],type="l",main="Death",xlab="Patient No.",
     ylab="CUSUM Statistic")
    lines(c(0,1e99), c(70,70),lty=2,col=2)
    lines(c(0,1e99), c(38,38),lty=2,col=3)

#########################################################################
# Run the CUSUM with thresholds as in Steiner et al. (1999).
# After each alarm the CUSUM statistic is set to zero and
# monitoring continues from this point. Triangles indicate alarm
# in the respective CUSUM (nearmiss or death). If in both
# simultaneously then an alarm is caued by the secondary limits.
#########################################################################
pb.surv2 <- pairedbinCUSUM(deleval,control=list(theta0=theta0,
             theta1=theta1,h1=70*w1,h2=32*w2,h11=38*w1,h22=17*w2))

plot(pb.surv2, xaxis.labelFormat=NULL)
```

permutationTest *Monte Carlo Permutation Test for Paired Individual Scores*

Description

As test statistic the difference between mean scores from model A and mean scores from model B is used. Under the null hypothesis of no difference, the actually observed difference between mean scores should not be notably different from the distribution of the test statistic under permutation. As the computation of all possible permutations is only feasible for small datasets, a random sample of permutations is used to obtain the null distribution. The resulting p-value thus depends on the .Random.seed.

Usage

```
permutationTest(score1, score2, nPermutation = 9999,
                plot = FALSE, verbose = FALSE)
```

Arguments

score1, score2	numeric vectors of scores to compare
nPermutation	number of random permutations to conduct
plot	logical indicating if a histogram of the nPermutation permutation test statistics should be plotted with a vertical line marking the observed difference of the means.
verbose	logical indicating if the results should be printed in one line.

Details

For each permutation, we first randomly assign the membership of the n individual scores to either model A or B with probability 0.5. We then compute the respective difference in mean for model A and B in this permuted set of scores. The Monte Carlo p-value is then given by (1 + #permuted differences larger than observed difference (in absolute value)) / (1 + nPermutation).

Value

a list of the following elements:

diffObs	observed difference in mean scores, i.e., mean(score1) - mean(score2)
pVal.permut	p-value of the permutation test
pVal.t	p-value of the corresponding t.test(score1, score2, paired=TRUE)

Author(s)

Michaela Paul

See Also

scores to obtain individual scores for oneStepAhead predictions from a model.

Package **coin** for a comprehensive permutation test framework, specifically its function symmetry_test to compare paired samples.

---

pit *Non-Randomized Version of the PIT Histogram (for Count Data)*

Description

See Czado et al. (2009).

Usage

```
pit(x, pdistr, J = 10, relative = TRUE, ..., plot = list())
```

Arguments

x	numeric vector representing the observed counts.
pdistr	either a list of predictive cumulative distribution functions for the observations x, or (the name of) a single predictive CDF used for all x (with potentially varying arguments ...). It is checked that the predictive CDF returns 0 at x=-1. The name of its first argument can be different from x, e.g., pdistr="pnbinom" is possible. If pdistr is a single function and no additional ... arguments are supplied, pdistr is assumed to be vectorized, i.e., it is simply called as pdistr(x) and pdistr(x-1). Otherwise, the predictive CDF is called sequentially and does not need to be vectorized.
J	the number of bins of the histogram.
relative	logical indicating if relative frequency or the density should be plotted.
...	ignored if pdistr is a list. Otherwise, such additional arguments are used in sequential calls of pdistr via mapply(pdistr, x, ...).
plot	a list of arguments for plot.histogram or NULL in which case no plot will be produced.

Value

an object of class "histogram" (see hist). It is returned invisibly if a plot is produced.

Author(s)

Michaela Paul and Sebastian Meyer

References

Czado, C., Gneiting, T. & Held, L. (2009): Predictive model assessment for count data. *Biometrics*, **65**, 1254-1261.

Examples

```
## Simulation example of Czado et al. (2009, Section 2.4)
set.seed(100)
x <- rnbinom(200, mu = 5, size = 2)
pdistrs <- list("NB(5,0)"   = function (x) ppois(x, lambda=5),
                "NB(5,1/2)" = function (x) pnbinom(x, mu=5, size=2),
                "NB(5,1)"   = function (x) pnbinom(x, mu=5, size=1))
## Reproduce Figure 1
op <- par(mfrow = c(1,3))
for (i in seq_along(pdistrs)) {
    pit(x, pdistr = pdistrs[[i]], J = 10, relative = TRUE,
        plot = list(ylim = c(0,2.75), main = names(pdistrs)[i]))
    box()
}
par(op)

## Alternative call using ... arguments for pdistr (less efficient)
stopifnot(identical(pit(x, "pnbinom", mu = 5, size = 2, plot = NULL),
                    pit(x, pdistrs[[2]], plot = NULL)))
```

---

plot.atwins *Plot results of a twins model fit*

Description

Plot results of fitting a twins model using MCMC output. Plots similar to those in the Held et al. (2006) paper are generated

Usage

```
  ## S3 method for class 'atwins'
plot(x, which=c(1,4,6,7), ask=TRUE, ...)
```

Arguments

x	An object of class atwins.
which	a vector containing the different plot types to show **1** A plot of the observed time series Z is shown together with posterior means for the number of endemic cases (X) and number of epidemic cases (Y). **2** This plot shows trace plots of the gamma parameters over all MCMC samples. **3** This shows a trace plot of psi, which controls the overdispersion in the model. **4** Autocorrelation functions for K and psi are shown in order to judge whether the MCMC sampler has converged.

5 Shows a plot of the posterior mean of the seasonal model nu[t] together with 95% credibility intervals based on the quantiles of the posterior.

6 Histograms illustrating the posterior density for K and psi. The first one corresponds to Fig. 4(f) in the paper.

7 Histograms illustrating the predictive posterior density for the next observed number of cases Z[n+1]. Compare with Fig.5 in the paper.

`ask` Boolean indicating whether to ask for a newline before showing the next plot.

`...` Additional control for the plots, which are currently ignored.

Details

For details see the plots in the paper. Basically MCMC output is visualized. This function is together with `algo.twins` still experimental.

Value

This function does not return anything.

Author(s)

M. Hofmann and M. Höhle

References

Held, L., Hofmann, M., Höhle, M. and Schmid V. (2006) A two-component model for counts of infectious diseases, Biostatistics, **7**, pp. 422–437.

See Also

algo.twins

Examples

```
## Not run:
#Apparently, the algo.atwins can crash on some LINUX systems
#thus for now the example section is commented

#Load the data used in the Held et al. (2006) paper
data("hepatitisA")

#Fix seed - this is used for the MCMC samplers in twins
set.seed(123)

#Call algorithm and save result
otwins <- algo.twins(hepatitisA)

#This shows the entire output
plot(otwins,which=c(1,2),ask=FALSE)

## End(Not run)
```

plot.disProg *Plot Generation of the Observed and the defined Outbreak States of a (multivariate) time series*

Description

Plotting of a disProg object.

Usage

```
  ## S3 method for class 'disProg'
plot(x, title = "", xaxis.years=TRUE, startyear = x$start[1],
   firstweek = x$start[2], as.one=TRUE, same.scale=TRUE, ...)
  ## S3 method for class 'disProg.one'
plot(x, title = "", xaxis.years=TRUE, quarters=TRUE,
  startyear =  x$start[1], firstweek = x$start[2], ylim=NULL, xlab="time",
   ylab="No. infected",type="hh",lty=c(1,1),col=c(1,1),
   outbreak.symbol = list(pch=3, col=3), legend.opts=list(x="top",
   legend=c("Infected", "Outbreak"), lty=NULL,pch=NULL,col=NULL), ...)
```

Arguments

x	object of class disProg
title	plot title
xaxis.years	if TRUE, the x axis is labeled using years
quarters	add quarters to the plot
startyear	year to begin the axis labeling (the year where the oldest data come from). This arguments will be obsolete in sts.
firstweek	number of the first week of January in the first year (just for axis labeling grounds)
as.one	if TRUE all individual time series are shown in one plot
same.scale	if TRUE all plots have same scale
ylim	range of y axis
xlab	label of the x-axis
ylab	label of the y-axis
type	line type of the observed counts (should be hh)
lty	line type of the observed counts
col	color of the observed count lines
outbreak.symbol	list with entries pch and col specifying the plot symbol
legend.opts	a list containing the entries to be sent to the legend function. If no legend is requested use legend.opts=NULL. Otherwise, the following arguments are default

`x top`

`legend` The names infected and outbreak

`lty` If `NULL` the `lty` argument will be used

`pch` If `NULL` the `pch` argument is used

`col` If `NULL` the `col` argument is used

An further arguments to the `legend` function are just provided as additional elements of this list, e.g. `horiz=TRUE`.

`...` further arguments for the function `matplot`

Value

`a plot` showing the number of infected and the defined alarm status for a time series created by simulation or given in data either in one single plot or in several plots for each individual time series.

Author(s)

M. Höhle with contributions by A. Riebler and C. Lang

Examples

```
# Plotting of simulated data
disProgObj <- sim.pointSource(p = 0.99, r = 0.5, length = 208,
                              A = 1, alpha = 1, beta = 0, phi = 0,
                              frequency = 1, state = NULL, K = 5)
# plot the simulated disease with the defined outbreaks
plot(disProgObj)
title <- "Number of Infected and Defined Outbreak Positions for Simulated Data"
plot(disProgObj, title = title)
plot(disProgObj, title = title, xaxis.years=TRUE,
        startyear = 1999, firstweek = 13)
plot(disProgObj, title = title, xaxis.years=TRUE,
        startyear = 1999, firstweek = 14)

# Plotting of measles data
data(measles.weser)
# one plot
plot(measles.weser, title = "measles cases in the district Weser-Ems",
        xaxis.years=TRUE, startyear= 2001, firstweek=1)
# plot cases for each "Kreis"
plot(measles.weser, same.scale=TRUE, as.one=FALSE)
```

plot.hhh4 *Plots for Fitted* hhh4*-models*

Description

There are five types of plots for fitted hhh4 models:

- Plot the "fitted" component means (of selected units) along time along with the observed counts.
- Plot the estimated "season"ality of the three components.
- Plot the time-course of the dominant eigenvalue "maxEV".
- If the units of the corresponding multivariate "sts" object represent different regions, a map of estimated region-specific intercepts ("ri") of a selected model component can be produced.
- Plot the (estimated) neighbourhood weights ("neweights") as a function of neighbourhood order (shortest-path distance between regions), i.e., w_ji ~ o_ji.

Usage

```
## S3 method for class 'hhh4'
plot(x, type=c("fitted", "season", "maxEV", "ri", "neweights"), ...)

plotHHH4_fitted(x, units = 1, names = NULL,
                col = c("orange", "blue", "grey85", "black"),
                pch = 19, pt.cex = 0.6,
                par.settings = list(),
                legend = TRUE, legend.args = list(),
                legend.observed = FALSE, ...)

plotHHH4_fitted1(x, unit = 1, main = NULL,
                 col = c("grey30", "grey60", "grey85", "grey0"),
                 pch = 19, pt.cex = 0.6, border = col,
                 start = x$stsObj@start, end = NULL,
                 xlim = NULL, ylim = NULL, xlab = "", ylab = "No. infected",
                 hide0s = FALSE, meanHHH = NULL)

plotHHH4_season(..., components = NULL, intercept = FALSE,
                xlim = NULL, ylim = NULL,
                xlab = NULL, ylab = "", main = NULL,
                par.settings = list(), matplot.args = list(),
                legend = NULL, legend.args = list(),
                refline.args = list(), unit = 1)
getMaxEV_season(x)

plotHHH4_maxEV(...,
               matplot.args = list(), refline.args = list(),
               legend.args = list())
```

```
getMaxEV(x)

plotHHH4_ri(x, component, labels = FALSE, sp.layout = NULL,
            gpar.missing = list(col = "darkgrey", lty = 2, lwd = 2),
            ...)

plotHHH4_neweights(x, plotter = boxplot, ...,
                   exclude = 0, maxlag = Inf)
```

Arguments

x	a fitted hhh4 object.
type	type of plot: either "fitted" component means of selected units along time along with the observed counts, or "season"ality plots of the model components and the epidemic dominant eigenvalue (which may also be plotted along overall time by type="maxEV", especially if the model contains time-varying neighbourhood weights or unit-specific epidemic effects), or a map of estimated region-specific random intercepts ("ri") of a specific model component. The latter requires the model x to contain the selected random intercepts, and x$stsObj to contain a map.
...	For plotHHH4_season and plotHHH4_maxEV, one or more hhh4-fits, or a single list of these. Otherwise further arguments passed on to other functions. For the plot-method these go to the specific plot type function. plotHHH4_fitted passes them to plotHHH4_fitted1, which is called sequentially for every unit in units. plotHHH4_ri passes additional arguments to spplot, and plotHHH4_neweights to the plotter.
units,unit	integer or character vector specifying a single unit or possibly multiple units to plot. It indexes colnames(x$stsObj). In the seasonality plot, selection of a unit is only relevant if the model contains unit-specific intercepts or seasonality terms.
names,main	main title(s) for the selected unit(s) / components. If NULL (default), plotHHH4_fitted1 will use the appropriate element of colnames(x$stsObj), whereas plotHHH4_season uses default titles.
col,border	length 4 vector specifying the colors for the spatio-temporal, autoregressive, endemic, and observed elements in this order. For plotHHH4_fitted, a length 3 vector also works, in which case the default "black" will be used for the dots of observed counts.
pch,pt.cex	style specifications for the dots drawn to represent the observed counts. pch=NA can be used to disable these dots.
par.settings	list of graphical parameters for par. Sensible defaults for mfrow, mar and las will be applied unless overridden or !is.list(par.settings).
legend	Integer vector specifying in which of the length(units) frames the legend should be drawn. If a logical vector is supplied, which(legend) determines the frame selection, i.e., the default is to drawn the legend in the first (upper left) frame only, and legend=FALSE results in no legend being drawn.

`legend.args` list of arguments for `legend`, e.g., to modify the default positioning `list(x="topright", inset=0.02)`.

`legend.observed` logical indicating if the legend should contain a line for the dots corresponding to observed counts.

`start,end` time range to plot specified by vectors of length two in the form `c(year,number)`, see `"sts"`.

`xlim` numeric vector of length 2 specifying the x-axis range. The default (`NULL`) is to plot the complete time range.

`ylim` y-axis range. For `type="fitted"`, this defaults to `c(0,max(observed(x$stsObj)[,unit]))`. For `type="season"`, `ylim` must be a list of length `length(components)` specifying the range for every component plot, or a named list to customize only a subset of these. If only one `ylim` is specified, it will be recycled for all `components` plots.

`xlab,ylab` axis labels. For `plotHHH4_season`, `ylab` specifies the y-axis labels for all `components` in a list (similar to `ylim`). If `NULL` or incomplete, default mathematical expressions are used. If a single name is supplied such as the default `ylab=""` (to omit y-axis labels), it is used for all `components`.

`hide0s` logical indicating if dots for zero observed counts should be omitted. Especially useful if there are too many.

`meanHHH` (internal) use different component means than those estimated and available from `x`.

`components` character vector of component names, i.e., a subset of `c("ar", "ne", "end")`, for which to plot the estimated seasonality. If `NULL` (the default), only components which appear in any of the models in `...` are plotted.
A seasonality plot of the epidemic dominant eigenvalue is also available by including `"maxEV"` in `components`, but it only supports models without epidemic covariates/offsets.

`intercept` logical indicating whether to plot seasonality as a multiplicative effect on the respective component (the default `intercept=FALSE`), or additionally multiplied by the corresponding intercept (`intercept=TRUE`). The latter only makes sense if there are no further (non-centered) covariates/offsets in the component.

`matplot.args` list of line style specifications passed to `matplot`, e.g., `lty`, `lwd`, `col`.

`refline.args` list of line style specifications passed to `abline` to draw the reference line in the plot of the dominant eigenvalue, e.g., `lty` and `col`.

`component` component for which to plot the estimated region-specific random intercepts. Must partially match one of `colnames(ranef(x, tomatrix=TRUE))`.

`labels` determines if and how regions are labeled, see `layout.labels`.

`sp.layout` optional list of additional layout items, see `spplot`.

`gpar.missing` list of graphical parameters applied to regions with missing random intercepts (i.e., not included in the model). Supplementary regions won't be plotted at all if `!is.list(gpar.missing)`.

`plotter` the (name of a) function used to produce the plot of weights (a numeric vector) as a function of neighbourhood order (a factor variable). It is called as `plotter(Weight ~ Distance, ...)` and defaults to `boxplot`. A useful alternative is, e.g., `stripplot` from package **lattice**.

exclude	vector of neighbourhood orders to be excluded from plotting (passed to factor). By default, the neighbourhood weight for order 0 is not shown, which is usually zero anyway.
maxlag	maximum order of neighbourhood to be assumed when computing the nbOrder matrix. This additional step is necessary iff neighbourhood(x$stsObj) only specifies a binary adjacency matrix.

Value

plotHHH4_fitted1 invisibly returns a matrix of the fitted component means for the selected unit, and plotHHH4_fitted returns these in a list for all units.
plotHHH4_season invisibly returns the plotted y-values, i.e. the multiplicative seasonality effect within each of components. Note that this will include the intercept, i.e. the point estimate of $exp(intercept + seasonality)$ is plotted and returned.
getMaxEV_season returns a list with elements "maxEV.season" (as plotted by plotHHH4_season(..., components="maxE "maxEV.const" and "Lambda.const" (the Lambda matrix and its dominant eigenvalue if time effects are ignored).
plotHHH4_maxEV (invisibly) and getMaxEV return the dominant eigenvalue of the Λ_t matrix for all time points t of x$stsObj.
plotHHH4_ri returns the generated spplot, which is a **lattice** plot of class "trellis".
plotHHH4_neweights eventually calls plotter and thus returns whatever is returned by that function.

Author(s)

Sebastian Meyer

References

Held, L. and Paul, M. (2012): Modeling seasonality in space-time infectious disease surveillance data. *Biometrical Journal*, **54**, 824-843.
DOI-Link: http://dx.doi.org/10.1002/bimj.201200037

See Also

other methods for hhh4 fits, e.g., summary.hhh4.

Examples

```
data("measlesWeserEms")

## fit a simple hhh4 model
measlesModel <- list(
    ar = list(f = ~ 1),
    end = list(f = addSeason2formula(~0 + ri(type="iid"), S=1, period=52),
               offset = population(measlesWeserEms)),
    family = "NegBin1"
    )
measlesFit <- hhh4(measlesWeserEms, measlesModel)
```

```
## fitted values for a single unit
plot(measlesFit, units=2)

## plot the multiplicative effect of seasonality
plot(measlesFit, type="season")

## dominant eigenvalue of the Lambda matrix (cf. Held and Paul, 2012)
getMaxEV(measlesFit)  # here simply constant and equal to exp(ar.1)
plot(measlesFit, type="maxEV")  # not very exciting

## random intercepts of the endemic component
plot(measlesFit, type="ri", component="end", labels=list(font=3, labels="GEN"))

## neighbourhood weights as a function of neighbourhood order
plot(measlesFit, type="neweights")  # boring, model has no "ne" component

## fitted values for the 6 regions with most cases and some customization
bigunits <- tail(names(sort(colSums(observed(measlesWeserEms)))), 6)
plot(measlesFit, units=bigunits,
     names=measlesWeserEms@map@data[bigunits,"GEN"],
     legend=5, legend.args=list(x="top"), xlab="Time (weekly)",
     hide0s=TRUE, ylim=c(0,max(observed(measlesWeserEms)[,bigunits])),
     start=c(2002,1), end=c(2002,26), par.settings=list(xaxs="i"))
```

`plot.survRes` *Plot a survRes object*

Description

Plotting of a (multivariate) `survRes` object. The function `plot.survRes.one` is used as a helper function to plot a univariate time series.

Usage

```
  ## S3 method for class 'survRes'
plot(x, method=x$control$name, disease=x$control$data,
  xaxis.years=TRUE,startyear = 2001, firstweek = 1, same.scale=TRUE,...)
  ## S3 method for class 'survRes.one'
plot(x, method=x$control$name, disease=x$control$data,
   domany=FALSE,ylim=NULL,xaxis.years=TRUE,startyear = 2001, firstweek = 1,
   xlab="time", ylab="No. infected", main=NULL, type="hhs",
   lty=c(1,1,2),col=c(1,1,4),
   outbreak.symbol = list(pch=3,col=3),alarm.symbol=list(pch=24,col=2),
   legend.opts=list(x="top",
   legend=c("Infected", "Upperbound", "Alarm", "Outbreak"),
   lty=NULL,col=NULL,pch=NULL), ...)
```

Arguments

x	object of class survRes
method	surveillance method to be used in title
disease	name of disease in title
xaxis.years	Boolean indicating whether to show a year based x-axis for weekly data
domany	Boolean telling the function whether it is called for a multivariate (TRUE) or univariate (FALSE) survRes object. In case of TRUE no titles are drawn.
ylim	range of y axis
startyear	year to begin the axis labeling (the year where the oldest data come from)
firstweek	number of the first week of January in the first year (just for axis labeling reasons)
xlab	label of the x-axis
ylab	label of the y-axis
main	the title of the graphics is generated from the method and disease arguments if not specified otherwise
same.scale	plot all time series with the same ylim? Defaults to true.
type	line type of the observed counts (first two elements) and the upper bound (third element)
lty	vector of size 3 speciying the line type of the observed counts (left, right) and the upperbound line
col	vector with three elements: color of left bar and color of top bar, color of right bar, col of the upperbound line.
outbreak.symbol	list with entries pch and col specifying the plot symbol
alarm.symbol	list with entries pch and col specifying the plot symbol
legend.opts	a list containing the entries to be sent to the legend function. If no legend is requested use legend.opts=NULL. Otherwise, the following arguments are default x top legend The names infected and outbreak. lty If NULL the lty argument will be used pch If NULL the pch argument is used col If NULL the col argument is used Any further arguments to the legend function are just provided as additional elements of this list, e.g. horiz=TRUE.
...	further arguments for the function matplot. If e.g. xlab or main are provided they overwrite the default values.

Details

The plot.survRes.one is intended for internal use. At the moment none of the surveillance methods support multivariate survRes objects. New versions of the packages currently under development will handle this.

Value

none. A plot showing the number of infected, the threshold for recognizing an outbreak, the alarm status and the outbreak status is generated.

Author(s)

M. Höhle

Examples

```
data(ha)
ctrl <- list(range = 209:290, b = 2, w = 6, alpha = 0.005)
plot(algo.bayes(aggregate(ha), control = ctrl))
```

---

poly2adjmat *Derive Adjacency Structure of* "SpatialPolygons"

Description

Wrapping around functionality of the **spdep** package, this function computes the symmetric, binary (0/1), adjacency matrix from a "SpatialPolygons" object. It essentially applies nb2mat(poly2nb(SpP, ...), style="B"

Usage

```
poly2adjmat(SpP, ..., zero.policy = TRUE)
```

Arguments

SpP	an object inheriting from "SpatialPolygons-class".
...	arguments passed to poly2nb.
zero.policy	logical indicating if islands are allowed, see nb2mat.

Value

a symmetric numeric indicator matrix of size length(SpP)^2 representing polygon adjacencies.

Author(s)

(of this wrapper) Sebastian Meyer

See Also

poly2nb in package **spdep**

Examples

```
if (requireNamespace("spdep")) {
    ## generate adjacency matrix for districts of Bayern and Baden-Wuerttemberg
    data("fluBYBW")
    adjmat <- poly2adjmat(fluBYBW@map)

    ## same as already stored in the neighbourhood slot (in different order)
    stopifnot(all.equal(adjmat,
                        neighbourhood(fluBYBW)[rownames(adjmat),colnames(adjmat)]))

    ## the neighbourhood graph can be plotted with spdep
    plot(spdep::mat2listw(adjmat), coordinates(fluBYBW@map))
}
```

`polyAtBorder` *Indicate Polygons at the Border*

Description

Determines which polygons of a "SpatialPolygons" object are at the border, i.e. have coordinates in common with the spatial union of all polygons (constructed using unionSpatialPolygons).

Usage

```
polyAtBorder(SpP, snap = sqrt(.Machine$double.eps),
             method = "rgeos", ...)
```

Arguments

SpP	an object of class "SpatialPolygons".
snap	tolerance used to consider coordinates as identical.
method	method to use for unionSpatialPolygons. Defaults to "rgeos", since **polyclip** uses integer arithmetic, which causes rounding errors usually requiring tuning of (i.e., increasing) the tolerance parameter snap (see example below).
...	further arguments passed to the chosen method.

Value

logical vector of the same length as SpP also inheriting its row.names.

Author(s)

Sebastian Meyer

Examples

```
## Load districts of Germany
load(system.file("shapes", "districtsD.RData", package="surveillance"))

## indicate districts at the border
districtAtBorder <- polyAtBorder(districtsD)

## plot to check
plot(districtsD, col=districtAtBorder)

## polyclip cannot be used with the default snapping tolerance
## but increasing the tolerance gives the correct result
stopifnot(identical(districtAtBorder,
                    polyAtBorder(districtsD, snap=1e-6, method="polyclip")))
```

`predict.ah` *Predictions from a HHH model*

Description

Use a ah or ahg object for prediction.

Usage

```
## S3 method for class 'ah'
predict(object,newdata=NULL,
   type=c("response","endemic","epi.own","epi.neighbours"), ...)
```

Arguments

`object`	object of class ah or ahg
`newdata`	optionally, a disProgObject with which to predict; if omitted, the fitted mean is returned.
`type`	the type of prediction required. The default is on the scale of the response variable (endemic and epidemic part). The alternative "endemic" returns only the endemic part (i.e. $n_{it}\nu_{it}$), "epi.own" and "epi.neighbours" return the epidemic part (i.e. $\lambda_i y_{i,t}$ and $\phi_i \sum_{j\sim i} y_{j,t-1}$)
`...`	not really used

Details

this function is still experimental

Value

matrix of values containing the mean μ_{it} for each region and time point.

primeFactors *Prime number factorization*

Description

Computes prime number factorization of an integer x.

Usage

```
primeFactors(x)
```

Arguments

x	an integer

Value

vector with prime number factorization of x

print.algoQV *Print quality value object*

Description

Print a single qualitity value object in a nicely formatted way

Usage

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

Arguments

x	Quality Values object generated with quality
...	Further arguments (not reall used)

Examples

```
# Create a test object
disProgObj <- sim.pointSource(p = 0.99, r = 0.5, length = 200, A = 1,
                                    alpha = 1, beta = 0, phi = 0,
                                    frequency = 1, state = NULL, K = 1.7)

# Let this object be tested from rki1
survResObj <- algo.rki1(disProgObj, control = list(range = 50:200))

# Compute the quality values in a nice formatted way
algo.quality(survResObj)
```

qlomax *Quantile Function of the Lomax Distribution*

Description

Quantile function of the Lomax distribution with positive scale parameter `scale` (often denoted as σ) and positive shape parameter `shape` (often denoted as α). This implementation does not include any checks, but only the raw formula `scale * ((1-p)^(-1/shape) - 1)`. Another implementation can be found as `qlomax` in the package **VGAM**.

Usage

```
qlomax(p, scale, shape)
```

Arguments

`p`	vector of probabilities.
`scale`	positive scale parameter.
`shape`	positive shape parameter.

Value

Numeric vector of quantiles corresponding to the probabilities `p`.

Author(s)

Sebastian Meyer

See Also

`Lomax` in package **VGAM**.

Examples

```
qlomax(0.99, 1, 2)
```

R0 *Computes basic reproduction numbers from fitted models*

Description

The S3 generic function R0 defined in package **surveillance** is intended to compute basic reproduction numbers from fitted epidemic models. The package currently defines a method for the "twinstim" class, which computes mean numbers of infections caused by infected individuals depending on the event type and marks attached to the individual, which contribute to the infection pressure in the epidemic predictor of that class. There is also a method for simulated "epidataCS" (just a wrapper for the "twinstim"-method).

Usage

```
R0(object, ...)

## S3 method for class 'twinstim'
R0(object, newevents, trimmed = TRUE, newcoef = NULL, ...)
## S3 method for class 'simEpidataCS'
R0(object, trimmed = TRUE, ...)
```

Arguments

object	A fitted epidemic model object for which an R0 method exists.
newevents	an optional data.frame of events for which the basic reproduction numbers should be calculated. If omitted, it is calculated for the original events from the fit. In this case, if trimmed = TRUE (the default), the result is just object$R0; however, if trimmed = FALSE, the model environment is required, i.e. object must have been fitted with model = TRUE. For the twinstim method, newevents must at least contain the following columns: the time of the events, the factor variable type, the interaction ranges eps.t and eps.s, as well as columns for the marks used in the epidemic component of the fitted twinstim object as stored in formula(object)$epidemic. The coding of the variables must of course be the same as used for fitting. For trimmed R0 values, newevents must additionally contain the components .influenceRegion and, if using the Fcircle trick in the siaf specification, also .bdist (cf. the hidden columns in the events component of class "epidataCS").
trimmed	logical indicating if the individual reproduction numbers should be calculated by integrating the epidemic intensities over the observation period and region only (trimmed = TRUE) or over the whole time-space domain R+ x R^2 (trimmed = FALSE). By default, if newevents is missing, the trimmed R0 values stored in object are returned. Trimming means that events near the (spatial or temporal) edges of the observation domain have lower reproduction numbers (ceteris paribus) because events outside the observation domain are not observed.
newcoef	the model parameters to use when calculating reproduction numbers. The default (NULL) is to use the MLE coef(object). This argument mainly serves the

construction of Monte Carlo confidence intervals by evaluating `R0` for parameter vectors sampled from the asymptotic multivariate normal distribution of the MLE, see Examples.

`...` additional arguments passed to methods. Currently unused for the `twinstim` method.

Details

For the "`twinstim`" class, the individual-specific mean number μ_j of infections caused by individual (event) j inside its theoretical (untrimmed) spatio-temporal range of interaction given by its `eps.t` (ϵ) and `eps.s` (δ) values is defined as follows (cf. Meyer et al, 2012):

$$\mu_j = e^{\eta_j} \cdot \int_0^{\epsilon} g(t)dt \cdot \int_{b(\mathbf{0},\delta)} f(\boldsymbol{s})d\boldsymbol{s}.$$

Here, $b(\mathbf{0}, \delta)$ denotes the disc centred at (0,0)' with radius δ, η_j is the epidemic linear predictor, $g(t)$ is the temporal interaction function, and $f(\boldsymbol{s})$ is the spatial interaction function. Alternatively, the `trimmed` (observed) mean reproduction numbers are obtain by integrating over the observed infectious domains of the individuals, i.e. integrate f over the intersection of the influence region with the observation region `W` (i.e. over $\{W \cap b(\boldsymbol{s}_j, \delta)\} - \boldsymbol{s}_j$) and g over the intersection of the observed infectious period with the observation period $(t_0; T]$ (i.e. over $(0; \min(T - t_j, \epsilon)]$).

(Numerical) Integration is performed exactly as during the fitting of `object`, for instance `object$control.siaf` is queried if necessary.

Value

numeric vector of estimated basic reproduction numbers from the fitted model `object` corresponding to the rows of `newevents` (if supplied) or the original fitted events including events of the prehistory.

Author(s)

Sebastian Meyer

References

Meyer, S., Elias, J. and Höhle, M. (2012): A space-time conditional intensity model for invasive meningococcal disease occurrence. *Biometrics*, **68**, 607-616.
DOI-Link: http://dx.doi.org/10.1111/j.1541-0420.2011.01684.x

Examples

```
## load the 'imdepi' data and a model fit
data("imdepi")
data("imdepifit")

## calculate individual and type-specific reproduction numbers
R0s <- R0(imdepifit)
tapply(R0s, imdepi$events@data[names(R0s), "type"], summary)
```

```
## untrimmed R0 for a specific event
R0(imdepifit, newevents=marks(imdepi)[1,], trimmed=FALSE)


### compute a Monte Carlo confidence interval

## use a simpler model with constant 'siaf' for speed
simplefit <- update(imdepifit, epidemic=~type, siaf=NULL)

## we'd like to compute the mean R0's by event type
meanR0ByType <- function (newcoef) {
    R0events <- R0(simplefit, newcoef=newcoef)
    tapply(R0events, imdepi$events@data[names(R0events),"type"], mean)
}
meansMLE <- meanR0ByType(newcoef=NULL)

## sample B times from asymptotic multivariate normal of the MLE
B <- 5  # CAVE: toy example! In practice this has to be much larger
set.seed(123)
parsamples <- MASS::mvrnorm(B, mu=coef(simplefit), Sigma=vcov(simplefit))

## for each sample compute the 'meanR0ByType'
meansMC <- apply(parsamples, 1, meanR0ByType)

## get the quantiles and print the result
cisMC <- apply(cbind(meansMLE, meansMC), 1, quantile, probs=c(0.025,0.975))
print(rbind(MLE=meansMLE, cisMC))
```

`readData` *Reading of Disease Data*

Description

Reading of disease data. In the package disease data are saved in a file `<abb>.txt` containing three columns – the weeknumber (week), the observed number of counts (observed) and a state (state). The data are read using `read.table(...,header=T)`, hence the file has to contain a header.

Usage

```
readData(abb,week53to52=TRUE,sysPath=TRUE)
```

Arguments

abb	abbreviation of the diseasename.
week53to52	Boolean indicating whether to convert RKI 53 Weeks System to 52 weeks a year
sysPath	Boolean, if TRUE then R automatically looks in the data directory of the **surveillance** package.

Details

This function is only kept for backwards compability. As of 0.9-2 all data should be read with `data`.

Value

`disProg`	a object `disProg` (disease progress) including a list of the observed and the state chain.

See Also

`m1`, `m2`, `m3`, `m4`, `m5`, `q1_nrwh`, `q2`, `s1`, `s2`, `s3`, `k1`, `n1`, `n2`, `h1_nrwrp`

Examples

```
readData("m5")

#To bring a single vector of counts into a format, which can be
#handled by readData. Assume ``counts'' is a vector of counts.
counts <- rpois(100,20)
counts <- data.frame("week"=1:length(counts),"observed"=counts,
                     "state"=rep(0,length(counts)))
write(c("week","observed","state"),file="disease.txt",ncol=3)
write(t(as.matrix(counts)),file="disease.txt",ncol=3,append=TRUE)
disease <- readData("disease",week53to52=FALSE,sysPath=FALSE)
```

`refvalIdxByDate` *Compute indices of reference value using Date class*

Description

The reference values are formed base on computatations of `seq` for Date class arguments.

Usage

```
refvalIdxByDate(t0, b, w, epochStr, epochs)
```

Arguments

`t0`	A Date object describing the time point
`b`	Number of years to go back in time
`w`	Half width of window to include reference values for
`epochStr`	One of `"1 month"`, `"1 week"` or `"1 day"`
`epochs`	Vector containing the epoch value of the sts/disProg object

Details

Using the Date class the reference values are formed as follows: Starting from `t0` go i, i= 1,...,`b` years back in time. For each year, go `w` epochs back and include from here to `w` epochs after `t0`.

In case of weeks we always go back to the closest monday of this date. In case of months we also go back in time to closest 1st of month.

Value

a vector of indices in epochs which match

`residuals.ah` *Residuals from a HHH model*

Description

Extracts model residuals from a `ah` or `ahg` object.

Usage

```
## S3 method for class 'ah'
residuals(object, type=c("deviance","pearson"), ...)
```

Arguments

`object`	object of class `ah` or `ahg`
`type`	the type of residuals which should be returned. The alternatives are "deviance" (default) and "pearson"
`...`	not really used

Details

this function is still experimental

Value

matrix with residuals for each region and time point.

`residualsCT` *Extract Cox-Snell-like Residuals of a Fitted Point Process*

Description

Extract the "residual process" (cf. Ogata, 1988) of a fitted point process modell specified through the conditional intensity function, for instance a model of class `"twinSIR"` or `"twinstim"`. The residuals are defined as the fitted cumulative intensities at the event times, and are generalized residuals similar to those discussed in Cox and Snell (1968).

Usage

```
## S3 method for class 'twinSIR'
residuals(object, ...)
## S3 method for class 'twinstim'
residuals(object, ...)
```

Arguments

`object`	an object of class `"twinSIR"` or `"twinstim"`.
`...`	unused (argument of the generic).

Details

For objects of class `twinstim`, the residuals may already be stored in the object as component `object$tau` if the model was fitted with `cumCIF = TRUE`. In this case, the `residuals` method just extracts these values. Otherwise, the residuals have to be calculated, which is only possible with access to the model environment, i.e. `object` must have been fitted with `model = TRUE`. The calulcated residuals are then also appended to `object` for future use. However, if `cumCIF` and `model` were both set to true in the `object` fit, then it is not possible to calculate the residuals and the method returns an error.

Value

Numeric vector of length the number of events of the corresponding point process fitted by `object`. This is the observed residual process.

Author(s)

Sebastian Meyer

References

Ogata, Y. (1988) Statistical models for earthquake occurrences and residual analysis for point processes. *Journal of the American Statistical Association*, 83, 9-27

Cox, D. R. & Snell, E. J. (1968) A general definition of residuals. *Journal of the Royal Statistical Society. Series B (Methodological)*, 30, 248-275

See Also

checkResidualProcess to graphically check the goodness-of-fit of the underlying model.

Examples

```
## Load the twinSIR() fit
data("foofit")
residuals(foofit)
## these residuals are, e.g., used by checkResidualProcess()
checkResidualProcess(foofit)
```

---

rotaBB *Rotavirus cases in Brandenburg, Germany, during 2002-2013 stratified by 5 age categories*

Description

Monthly reported number of rotavirus infections in the federal state of Brandenburg stratified by five age categories (00-04, 05-09, 10-14, 15-69, 70+) during 2002-2013.

Usage

```
data(rotaBB)
```

Format

A sts object.

Source

The data are queried on 19 Feb 2014 from the Survstat@RKI database of the German Robert Koch Institute (http://www3.rki.de/SurvStat/).

Examples

```
data(rotaBB)
```

`runifdisc` *Sample Points Uniformly on a Disc*

Description

Sample `n` points uniformly on a disc of radius `r` in two-dimensional euclidean space via transformation to polar coordinates: the angle is sampled uniformly from $U(0, 2\pi)$, the length is sampled uniformly from $\sqrt{U(0, r^2)}$. The sampled polar coordinates are then back-transformed to cartesian coordinates.

Usage

```
runifdisc(n, r = 1, buffer = 0)
```

Arguments

`n`	integer size of the sample.
`r`	numeric radius of the disc (centered at (0,0)).
`buffer`	radius of inner buffer zone without points.

Value

A two-column coordinate matrix of the sampled points.

Author(s)

Sebastian Meyer

See Also

`runifdisc` in package **spatstat**, which is slightly more flexible and integrated within the `"ppp"` class.

Examples

```
x <- surveillance:::runifdisc(1000, 3)
plot(x)
```

salmHospitalized *Hospitalized Salmonella cases in Germany 2004-2014*

Description

Reported number of cases of Salmonella in Germany 2004-2014 (early 2014) that were hospitalized. The corresponding total number of cases is indicated in the `slot populationFrac`. `multinomialTS` is `TRUE`.

Usage

```
data(salmNewport)
```

Format

A `sts` object.

Source

The data are queried from the Survstat@RKI database of the German Robert Koch Institute (http://www3.rki.de/SurvStat/).

Examples

```
data(salmHospitalized)
```

salmNewport *Salmonella Newport cases in Germany 2004-2013*

Description

Reported number of cases of the Salmonella Newport serovar in the 16 German federal states 2004-2013.

Usage

```
data(salmNewport)
```

Format

A `sts` object.

Source

The data are queried from the Survstat@RKI database of the German Robert Koch Institute (http://www3.rki.de/SurvStat/). A detailed description of the 2011 outbreak can be found in the publication

Bayer, C., Bernard, H., Prager, R., Rabsch, W., Hiller, P., Malorny, B., Pfefferkorn, B., Frank, C., de Jong, A., Friesema, I., Start, K., Rosner, B.M. (2014), An outbreak of Salmonella Newport associated with mung bean sprouts in Germany and the Netherlands, October to November 2011, Eurosurveillance 19(1):pii=20665.

Examples

```
data(salmNewport)
```

---

salmonella.agona *Salmonella Agona cases in the UK 1990-1995*

Description

Reported number of cases of the Salmonella Agona serovar in the UK 1990-1995. Note however that the counts do not correspond exactly to the ones used by Farrington et. al (1996).

Usage

```
data(salmonella.agona)
```

Format

A `disProg` object with 312 observations starting from week 1 in 1990.

Source

A statistical algorithm for the early detection of outbreaks of infectious disease, Farrington, C.P., Andrews, N.J, Beale A.D. and Catchpole, M.A. (1996). , J. R. Statist. Soc. A, 159, 547-563.

Examples

```
data(salmonella.agona)
```

scale.gpc.poly *Centering and Scaling a "gpc.poly" Polygon*

Description

This is a re-implementation of the corresponding method from package **gpclib** to also allow centering.

Usage

```
## S3 method for class 'gpc.poly'
scale(x, center = c(0,0), scale = c(1,1))
```

Arguments

x	an object of class "gpc.poly".
center	numeric vector of length 2 (x,y), which will be subtracted from the respective coordinates of x.
scale	numeric vector of length 2 (x,y), which serves as the divisor for the respective coordinates of x.

Value

A "gpc.poly", the shifted and/or scaled version of x.

shadar *Salmonella Hadar cases in Germany 2001-2006*

Description

Number of salmonella hadar cases in Germany 2001-2006. An increase is seen during 2006

Usage

```
data(shadar)
```

Format

A disProg object containing 295×1 observations starting from week 1 in 2001 to week 35 in 2006.

Source

Robert Koch-Institut: SurvStat: http://www3.rki.de/SurvStat; Queried on September 2006.

Robert Koch Institut, Epidemiologisches Bulletin 31/2006.

Examples

```
data(shadar)
plot(shadar)
```

`sim.pointSource` *Generation of Simulated Point Source Epidemy*

Description

Simulation of epidemies which were introduced by point sources. The basis of this proagramme is a combination of a Hidden Markov Modell (to get random timepoints for outbreaks) and a simple model (compare `sim.seasonalNoise`) to simulate the epidemy.

Usage

```
sim.pointSource(p = 0.99, r = 0.01, length = 400, A = 1,
    alpha = 1, beta = 0, phi = 0, frequency = 1, state = NULL, K)
```

Arguments

`p`	probability to get a new epidemy at time i if there was one at time i-1, default 0.99.
`r`	probability to get no new epidemy at time i if there was none at time i-1, default 0.01.
`length`	number of weeks to model, default 400. `length` is ignored if `state` is given. In this case the length of `state` is used.
`A`	amplitude (range of sinus), default = 1.
`alpha`	parameter to move along the y-axis (negative values not allowed) with alpha >= A, default = 1.
`beta`	regression coefficient, default = 0.
`phi`	factor to create seasonal moves (moves the curve along the x-axis), default = 0.
`frequency`	factor to determine the oscillation-frequency, default = 1.
`state`	use a state chain to define the status at this timepoint (outbreak or not). If not given a Markov chain is generated by the programme, default NULL.
`K`	additional weigth for an outbreak which influences the distribution parameter mu, default = 0.

Value

`disProg`	a object `disProg` (disease progress) including a list of the observed, the state chain and nearly all input parameters.

Author(s)

M. Höhle, A. Riebler, C. Lang

See Also

sim.seasonalNoise

Examples

```
# Plotting of simulated data
disProgObj <- sim.pointSource(p = 0.99, r = 0.5, length = 208,
                              A = 1, alpha = 1, beta = 0, phi = 0,
                              frequency = 1, state = NULL, K = 2)
# plot the simulated disease with the defined outbreaks
plot(disProgObj)

state <- rep(c(0,0,0,0,0,0,0,0,1,1), 20)
disProgObj <- sim.pointSource(state = state, K = 1.2)
plot(disProgObj)
```

---

sim.seasonalNoise *Generation of Background Noise for Simulated Timeseries*

Description

Generation of a cyclic model of a Poisson distribution as background data for a simulated timevector.

The mean of the Poisson distribution is modelled as:

$$\mu = \exp(A \sin(frequency \cdot \omega \cdot (t + \phi)) + \alpha + \beta * t + K * state)$$

Usage

```
sim.seasonalNoise(A = 1, alpha = 1, beta = 0, phi = 0,
                  length, frequency = 1, state = NULL, K = 0)
```

Arguments

A	amplitude (range of sinus), default = 1.
alpha	parameter to move along the y-axis (negative values not allowed) with alpha >= A, default = 1.
beta	regression coefficient, default = 0.
phi	factor to create seasonal moves (moves the curve along the x-axis), default = 0.
length	number of weeks to model.
frequency	factor to determine the oscillation-frequency, default = 1.
state	if a state chain is entered the outbreaks will be additional weighted by K.
K	additional weigth for an outbreak which influences the distribution parameter mu, default = 0.

Value

seasonNoise	Object of class seasonNoise which includes the modelled timevector, the parameter mu and all input parameters.

Author(s)

M. Höhle, A. Riebler, C. Lang

See Also

sim.pointSource

Examples

```
season <- sim.seasonalNoise(length = 300)
plot(season$seasonalBackground,type = "l")

# use a negative timetrend beta
season <- sim.seasonalNoise(beta = -0.003, length = 300)
plot(season$seasonalBackground,type = "l")
```

---

simHHH *Simulates data based on the model proposed by Held et. al (2005)*

Description

Simulates a multivariate time series of counts based on the Poisson/Negative Binomial model as described in Held et al. (2005).

Usage

```
## Default S3 method:
simHHH(model=NULL, control = list(coefs = list(alpha=1, gamma = 0, delta = 0,
       lambda = 0, phi = NULL, psi = NULL, period = 52),
       neighbourhood = NULL, population = NULL, start = NULL),

       length)

## S3 method for class 'ah'
simHHH(model, control = model$control, length)
```

Arguments

`control` list with

- **coefs** list with the following parameters of the model - if not specified, those parameters are omitted
  - **alpha** vector of length `m` with intercepts for `m` units or geographic areas respectively
  - **gamma** vector with parameters for the "sine" part of $\nu_{i,t}$
  - **delta** vector with parameters for the "cosine" part of $\nu_{i,t}$
  - **lambda** autoregressive parameter
  - **phi** autoregressive parameter for adjacent units
  - **psi** overdispersion parameter of the negative binomial model; `NULL` corresponds to a Poisson model
  - **period** period of the seasonal component, defaults to 52 for weekly data
- **neighbourhood** neighbourhood matrix of size $m \times m$ with element 1 if two units are adjacent; the default `NULL` assumes that there are no neighbours
- **population** matrix with population proportions; the default `NULL` sets $n_{i,t} = 1$
- **start** if `NULL`, the means of the endemic part in the `m` units is used as initial values $y_{i,0}$

`model` Result of a model fit with `algo.hhh`, the estimated parameters are used to simulate data

`length` number of time points to simulate

Details

Simulates data from a Poisson or a Negative Binomial model with mean

$$\mu_{it} = \lambda y_{i,t-1} + \phi \sum_{j \sim i} y_{j,t-1} + n_{it}\nu_{it}$$

where

$$\log \nu_{it} = \alpha_i + \sum_{s=1}^{S} (\gamma_s sin(\omega_s t) + \delta_s cos(\omega_s t))$$

$\omega_s = 2s\pi/$`period` are Fourier frequencies and n_{it} are possibly standardized population sizes.

Value

Returns a list with elements

`data` `disProgObj` of simulated data

`mean` matrix with mean $\mu_{i,t}$ that was used to simulate the data

`endemic` matrix with only the endemic part $\nu_{i,t}$

`coefs` list with parameters of the model

Note

The model does not contain a linear trend.

Source

Held, L., Höhle, M., Hofmann, M. (2005). A statistical framework for the analysis of multivariate infectious disease surveillance counts. Statistical Modelling, 5, p. 187-199.

`stcd` *Spatio-temporal cluster detection*

Description

Shiryaev-Roberts based prospective spatio-temporal cluster detection as in Assuncao & Correa (2009).

Usage

```
stcd(x, y,t,radius,epsilon,areaA, areaAcapBk, threshold, cusum=FALSE)
```

Arguments

`x`	Vector containing spatial x coordinate of the events.
`y`	Vector containing spatial y coordinate of the events.
`t`	Vector containing the time points of the events. It is assumed that the vector is sorted (early->last).
`radius`	Radius of the cluster to detect.
`epsilon`	Relative change of event-intensity within the cluster to detect. See reference paper for an explicit definition.
`areaA`	Area of the observation region A (single number) – This argument is currently ignored!
`areaAcapBk`	Area of A \ B(s_k,rho) for all k=1,...,n (vector). This argument is currently ignored!
`threshold`	Threshold limit for the alarm and should be equal to the desired Average-Run-Length (ARL) of the detector.
`cusum`	(logical) If `FALSE` (default) then the Shiryaev-Roberts detector is used as in the original article by Assuncao & Correa (2009), i.e. $R_n = \sum_{k=1}^{n} \Lambda_{k,n}$, where $\Lambda_{k,n}$ denotes the likelihood ratio between the in-control and out-of control model. If `TRUE`, CUSUM test statistic is used instead. Here, $$R_n = \max_{1 \leq k \leq n} \Lambda_{k,n}$$. Note that this has implications on what threshold will sound the alarm (CUSUM threshold needs to be smaller).

Details

Shiryaev-Roberts based spatio-temporal cluster detection based on the work in Assuncao and Correa (2009). The implementation is based on C++ code originally written by Marcos Oliveira Prates, UFMG, Brazil and provided by Thais Correa, UFMG, Brazil during her research stay in Munich. This stay was financially supported by the Munich Center of Health Sciences.

Note that the vectors `x`, `y` and `t` need to be of the same length. Furthermore, the vector `t` needs to be sorted (to improve speed, the latter is not verified within the function).

The current implementation uses a call to a C++ function to perform the actual computations of the test statistic. The function is currently experimental – data type and results may be subject to changes.

Value

A list with three components

`R`	A vector of the same length as the input containing the value of the test statistic for each observation.
`idxFA`	Index in the x,y,t vector causing a possible alarm. If no cluster was detected, then a value of `-1` is returned here.
`idxCC`	index in the x,y,t vector of the event containing the cluster. If no cluster was detected, then a value of `-1` is returned here.

Author(s)

M. O. Prates, T. Correa and M. Höhle

References

Assuncao, R. and Correa, T. (2009), Surveillance to detect emerging space-time clusters, Computational Statistics & Data Analysis, 53(8):2817-2830.

Examples

```
if (require("splancs")) {
    # load the data from package "splancs"
    data(burkitt, package="splancs")

    # order the times
    burkitt <- burkitt[order(burkitt$t), ]

    #Parameters for the SR detection
    epsilon <- 0.5 # relative change within the cluster
    radius <- 20 # radius
    threshold <- 161 # threshold limit

    res <- stcd(x=burkitt$x,
                y=burkitt$y,
                t=burkitt$t,
                radius=radius,
```

```
                epsilon=epsilon,
                areaA=1,
                areaAcapBk=1,
                threshold=threshold)

    #Index of the event
    which.max(res$R >= threshold)
}
```

---

sts-class *Class* "sts" *– surveillance time series*

Description

This is a lightweight S4 class to implement multivariate time series of counts used for public health surveillance data. The class captures the time series data as well as the spatial layout of the regions, where the data originate from.

Slots

epoch: Object of class numeric or specifying the time of observation. In old versions of the package this used to be the week numbers. However, depending on the freq argument, it can now be day or month as well. Furthermore, if epochAsDate=TRUE then it is the as.numeric representation of Date objects giving the exact date of the observation. Note: This slot used to be called week in earlier versions of the package, but has now been renamed to reflect the greater flexibility in the choice of observation time.

freq: If weekly data freq corresponds to 52, in case of monthly data freq is 12.

start: vector of length two denoting the year and the sample number (week, month, etc.) of the first observation

observed: A matrix of size length(epoch) times the number of regions containing the weekly/monthly number of counts in each region. The colnames of the matrix should match the ID values of the shapes in the map slot.

state: Matrix with the same dimension as observed containing booleans whether at the specific time point there was an outbreak in the region

alarm: Matrix with the same dimension as observed specifying whether an outbreak detection algorithm declared a specific time point in the region as having an alarm. If the object containins just observations then this slot is null.

upperbound: Matrix with upper bound values

neighbourhood: Symmetric matrix of size $(numberofregions)^2$ describing the neighbourhood structure. It may either be a binary adjacency matrix or contain neighbourhood orders.

populationFrac: A matrix of population fractions (with dimensions dim(observed)).

map: Object of class SpatialPolygonsDataFrame providing a shape of the areas which are monitored.

control: Object of class list, this is a rather free data type to be returned by the surveillance algorithms.

`epochAsDate`**:** Object of class `"logical"` stating whether to use a ISO 8601 representation of the `epoch` slot using the `Date` class (`epochAsDate=TRUE`) or just to interpret the epochs as numerics (`epochAsDate=FALSE`).

`multinomialTS`**:** Object of class `"logical"` stating whether to interpret the object as `observed` out of `population`, i.e. a multinomial interpretation instead of a count interpretation.

Methods

nrow `signature(x = "sts")`: extract number of rows of the `observed` matrix slot. The dimension of the other matrix slots is similar.

ncol `signature(x = "sts")`: extract number of columns of the `observed` matrix slot.

dim `signature(x = "sts")`: extract matrix dimensions of `observed` using `dim`.

observed `signature(x = "sts")`: extract the `observed` slot of an `sts` object.

population `signature(x = "sts")`: extract the `population` slot of an `sts` object.

multinomialTS `signature(x = "sts")`: extract the `multinomialTS` slot of an `sts` object.

neighbourhood `signature(x = "sts")`: extract the `neighbourhood` slot of an `sts` object.

alarms `signature(x = "sts")`: extract the `alarm` slot of an `sts` object.

upperbound `signature(x = "sts")`: extract the upperbound slot of an `sts` object.

control `signature(x = "sts")`: extract the `control` slot of an `sts` object.

epoch `signature(x = "sts")`: extract the `epoch` slot of an `sts` object. If ISO dates are used then the returned object is of class `Date`.

epochInYear `signature(x = "sts")`: Returns the epoch number within the year of the `epoch` slot.

colnames `signature(x="sts",do.NULL="missing",prefix="missing")`: extract `colnames` of the `observed` matrix.

initialize `signature(x="sts")`: the internal function `init.sts` is called, which assigns all slots.

aggregate `signature(x="sts")`: see `aggregate,sts-method`

year `signature(x = "sts")`: extracts the corresponding year of each observation of `x`

obsinyear `signature(x = "sts")`: extracts the corresponding week number within the year of each observation of `x`

as.data.frame `signature(x = "sts")`: converts the `observed`, `epoch`, `state` and `alarm` slots of `x` into a data frame with column names matching the colnames of the respective slots. Useful when one wants to fit a model based on the object

plot `signature(x="sts",y="missing")`: this method is the entry point to a collection of plot variants. It is also the successor of the `plot.disProg` and `plot.survRes` functions. The type of plot is specified using a formula `type`. See `stsplot` for details.

Author(s)

M. Höhle

Examples

```
if (requireNamespace("maptools")) {
   # load disProg-object "ha" and convert to S4-class "sts"
   data("ha")
   shpfile <- system.file("shapes/berlin.shp",package="surveillance")
   ha.sts <- disProg2sts(ha, map=maptools::readShapePoly(shpfile,IDvar="SNAME"))
} else {
   data("ha.sts")
   # is almost identical to the above except that German umlauts
   # have been replaced in 'ha.sts@map@data$BEZIRK' for compatibility reasons
}

ha.sts
plot(ha.sts, type=observed ~ 1 | unit)

#Convert ts/mts object to sts
z <- ts(matrix(rpois(300,10), 100, 3), start = c(1961, 1), frequency = 12)
sts.z <- as(z,"sts")
plot(sts.z)
```

`stsBP-class` *Class "stsBP" – a class inheriting from class* `sts` *which allows the user to store the results of back-projecting or nowcasting surveillance time series*

Description

A class inheriting from class `sts`, but with additional slots to store the result and associated confidence intervals from back projection of a `sts` object.

Slots

The slots are as for "`sts`". However, two additional slots exists.

`ci`**:** An array containing the upper and lower limit of the confidence interval.

`lambda`**:** Back projection component

Methods

The methods are the same as for "`sts`".

- `signature(from = "sts", to = "stsBP")` Convert an object of class `sts` to class `stsBP`.

Author(s)

M. Höhle

stsNC-class *Class "stsNC" – a class inheriting from class* sts *which allows the user to store the results of back-projecting surveillance time series*

Description

A class inheriting from class sts, but with additional slots to store the results of nowcasting.

Slots

The slots are as for "sts". However, a number of additional slots exists.

reportingTriangle: An array containing the upper and lower limit of the confidence interval.

predPMF: Predictive distribution for each nowcasted time point.

pi: A prediction interval for each nowcasted time point. This is calculated based on predPMF.

truth: An object of type sts containing the true number of cases.

delayCDF: List with the CDF of the estimated delay distribution for each method.

SR: Possible output of proper scoring rules

Methods

The methods are the same as for "sts".

- signature(from = "sts", to = "stsNC") Convert an object of class sts to class stsNC.
- reportingTrianglesignature(x = "stsNC"): extract the reportingTriangle slot of an stsNC object.
- delayCDFsignature(x = "stsNC"): extract the delayCDF slot of an stsNC object.
- scoresignature(x = "stsNC"): extract the scoring rules result slot of an stsNC object.

Author(s)

M. Höhle

stsplot *Plot-Methods for Surveillance Time-Series Objects*

Description

This page gives an overview of plot types which can be produced from objects of class "sts".

Usage

```
  ## S4 method for signature 'sts,missing'
plot(x, type = observed ~ time | unit, ...)
```

Arguments

x	an object of class "sts".
type	see Details.
...	arguments passed to the type-specific plot function.

Details

There are various types of plots which can be produced from an "sts" object. The type argument specifies the desired plot as a formula, which defaults to observed ~ time | unit, i.e., plot the time series of each unit separately. Arguments to specific plot functions can be passed as further arguments (...). The following list describes the plot variants:

observed ~ time | unit The default type shows ncol(x) plots with each containing the time series of one observational unit. The actual plotting per unit is done by the function stsplot_time1, called sequentially from stsplot_time.

observed ~ time The observations in x are aggregated over units and the resulting univariate time-series is plotted. The plotting is done by the function stsplot_time, which takes the same arguments as the old plot.survRes function.

alarm ~ time Generates a so called alarmplot for a multivariate sts object. For each time point and each series it is shown whether there is an alarm. In case of hierarchical surveillance the user can pass an additional argument lvl, which is a vector of the same length as rows in x specifying for each time series its level.

observed ~ unit produces a map of counts (or incidence) per region aggregated over time. See stsplot_space for optional arguments, details and examples.

observed ~ 1 | unit old version of the map plot, which supports shading regions with an alarm. The plotting is done by the function stsplot_spacetime. Use type=observed~unit for the new implementation as function stsplot_space (without alarm support, though).

observed ~ 1 | unit * time old version for animated maps via the stsplot_spacetime function. Each of the nrow(x) frames contains the number of counts per region for the current row in the observed matrix. It is possible to redirect the output into files, e.g. to generate an animated GIF. NOTE: the new animate.sts method supersedes this plot type!

Value

NULL (invisibly). The methods are called for their side-effects.

See Also

the documentation of the individual plot types stsplot_time, stsplot_space, stsplot_spacetime (obsolete), as well as the animate-method animate.sts. plot.survRes is the old implementation.

---

stsplot_space *Map of Disease Incidence During a Given Period*

Description

This is the plot variant of type=observed~unit for "sts" objects, i.e., plot(stsObj, type=observed~unit, ...) calls the function documented below. It produces an spplot where regions are color-coded according to disease incidence (either absolute counts or relative to population) during a given time period.

Usage

```
stsplot_space(x, tps = NULL, map = x@map, population = NULL,
              main = NULL, labels = FALSE, at = 10, col.regions = NULL,
              colorkey = list(space = "bottom", labels = list(at=at)),
              total.args = NULL, sp.layout = NULL, ...)
```

Arguments

x	an object of class "sts" or a matrix of counts, i.e., observed(stsObj), where especially colnames(x) have to be contained in row.names(map). If a matrix, the map object has to be provided explicitly. The possibility of specifying a matrix is, e.g., useful to plot mean counts of simulations from simulate.hhh4.
tps	a numeric vector of one or more time points. The unit-specific *sum* over all time points tps is plotted. The default tps=NULL means cumulation over the whole time period 1:nrow(x).
map	an object inheriting from "SpatialPolygons" representing the ncol(x) regions. By default the map slot of x is queried (which might be empty and is not applicable if x is a matrix of counts).
population	an optional numeric vector of population numbers in the ncol(x) regions. If given, incidence values instead of absolute counts are plotted.
main	a main title for the plot. If NULL and x is of class "sts", the time range of tps is put as the main title.
labels	determines if and how regions are labeled, see layout.labels.
at	either a number of levels (default: 10) for the categorization (color-coding) of counts, or specific break points to use, or a named list of a number of levels ("n"), a transformer ("trafo") of class "trans" defined by package **scales**, and

	optional further arguments for pretty. The default is the square root transformation (sqrt_trans). Note that the intervals given by at are closed on the left and open to the right, i.e., if specifying at manually as a vector of break points, make sure that max(at) is larger than the maximum observed count.
col.regions	a vector of fill colors of length length(at)-1.
colorkey	a list describing the color key, see levelplot. The default list elements will be updated by the provided list using modifyList.
total.args	an optional list of arguments for grid.text to have the overall number/incidence of cases printed at an edge of the map. The default settings are list(label="Overall: ", x=1, y=0), and total.args=list() will use all of them.
sp.layout	optional list of additional layout items, see spplot.
...	further arguments for spplot.

Value

a lattice plot of class "trellis", but see spplot.

Author(s)

Sebastian Meyer

See Also

the central stsplot-documentation for an overview of plot types, and animate.sts for animations of "sts" objects.

Examples

```
data("measlesWeserEms")

# default plot: total region-specific counts over all weeks
plot(measlesWeserEms, type=observed~unit)

# compare with old implementation
plot(measlesWeserEms, type=observed~1|unit)

# plot incidence with region labels
plot(measlesWeserEms, type=observed~unit,
     population=measlesWeserEms@map$POPULATION / 100000,
     labels=list(labels="GEN", cex=0.7, font=3))

# counts in the first week of the second year only (+ display overall)
plot(measlesWeserEms, type=observed~unit, tps=53, total.args=list())
```

stsplot_spacetime *Map of Disease Incidence*

Description

For each period (row) or for the overall period of the observed matrix of the "sts" object, a map showing the counts by region is produced. It is possible to redirect the output into files, e.g. to generate an animated GIF.

Usage

```
stsplot_spacetime(x, type, legend = NULL, opts.col = NULL, labels = TRUE,
                  wait.ms = 250, cex.lab = 0.7, verbose = FALSE,
                  dev.printer = NULL, ...)
```

Arguments

x	an object of class "sts".
type	a formula (see stsplot). For a map aggregated over time (no animation), use observed ~ 1 | unit, otherwise observed ~ 1 | unit * time.
legend	An object of type list containing the following items used for coloring • dxposition increments in x direction • dyposition increments in y direction • xposition in x • yposition in y • onceBoolean - if TRUE then only shown once If NULL then a default legend is used.
opts.col	A list containing the two elements • ncolorsNumber of colors to use for plotting • use.colorBoolean if TRUE then colors will be used in the palette, otherwise grayscale
labels	Boolean whether to add labels
wait.ms	Number of milliseconds to wait between each plot
cex.lab	cex of the labels
verbose	Boolean whether to write out extra information
dev.printer	Either NULL, which means that plotting is only to the screen otherwise a list with elements device, extension, width and height. This options is more or less obsolete - nowadays it's better to us the animation package to generate output to files.
...	Extra arguments sent to the plot function.

Note

The animate.sts method provides a re-implementation and supersedes this function!

Author(s)

Michael HÃ¶hle

See Also

Other stsplot types, and animate.sts for the new implementation.

Examples

```
data("ha.sts")
print(ha.sts)

## map of total counts by district
plot(ha.sts, type=observed ~ 1 | unit)
## only show a sub-period total for two selected districts
plot(ha.sts[1:20,1:2], type=observed ~ 1 | unit)

## Not run:
# space-time animation
plot(aggregate(ha.sts,nfreq=13), type= observed ~ 1 | unit * time)

#Configure a png device printer to save the frames
dev.printer <- list(device=png, extension=".png", width=640, height=480,
                    name=file.path(tempdir(),"berlin"))

#Do the animation (without extra sleeping time between frames)
plot(aggregate(ha.sts,nfreq=13), type = observed ~ 1 | unit * time,
     wait.ms=0, dev.printer=dev.printer)

#Use ImageMagick (you might have to adjust the path to 'convert')
system(paste("convert -delay 50 ",dev.printer$name,
           "*.png ", dev.printer$name, "-animated.gif",sep=""))

## End(Not run)
```

---

stsplot_time *Time-Series Plots for "sts" Objects*

Description

These are the plot variants of type=observed~time|unit, type=observed~time, and type=alarm~time for "sts" objects (see the central "sts" plot-method for an overview of plot types).

Usage

```
stsplot_time(x, units, method=x@control$name, disease=x@control$data,
             as.one=FALSE, same.scale=TRUE, par.list=list(), ...)

stsplot_time1(x, k=1, ylim=NULL,
              axes=TRUE, xaxis.tickFreq=list("%Q"=atChange),
              xaxis.labelFreq=xaxis.tickFreq, xaxis.labelFormat="%G\n\n%OQ",
              epochsAsDate=x@epochAsDate,
              xlab="time", ylab="No. infected", main=NULL,
              type="s", lty=c(1,1,2), col=c(NA,1,4), lwd=c(1,1,1),
              outbreak.symbol=list(pch=3, col=3, cex=1, lwd=1),
              alarm.symbol=list(pch=24, col=2, cex=1, lwd=1),
              legend.opts=list(x="top",legend=NULL,lty=NULL,pch=NULL,col=NULL),
              dx.upperbound=0L, hookFunc=function(){},
              .hookFuncInheritance=function() {}, ...)

stsplot_alarm(x, lvl=rep(1,nrow(x)), ylim=NULL,
              xaxis.tickFreq=list("%Q"=atChange),
              xaxis.labelFreq=xaxis.tickFreq, xaxis.labelFormat="%G\n\n%OQ",
              epochsAsDate=x@epochAsDate, xlab="time", main=NULL,
              type="hhs", lty=c(1,1,2), col=c(1,1,4),
              outbreak.symbol=list(pch=3, col=3, cex=1, lwd=1),
              alarm.symbol=list(pch=24, col=2, cex=1, lwd=1),
              cex=1, cex.yaxis=1, ...)
```

Arguments

x	an object of class "sts".
units	optional integer vector to select the units (=columns of observed(x)) to plot. stsplot_time1 is called for k in units. The default is to plot all time series.
method	name of the surveillance method to be used in the main title. Currently ignored.
disease	name of the disease to be used in the main title. Currently ignored.
as.one	logical indicating if all time series should be plotted within the same frame. This is currently not implemented for the "sts" class, but see plot.disProg and sts2disProg.
same.scale	logical indicating if all time series should be plotted with the same ylim. Default is to do so. Only relevant for multivariate plots (ncol(x) > 1).
par.list	a list of arguments delivered to a call of par before each plot.
k	the unit to plot, i.e., an element of 1:ncol(x).
ylim	the y limits of the plot(s). Ignored if same.scale=FALSE.
axes	a logical value indicating whether both axes should be drawn on the plot.
xaxis.tickFreq,xaxis.labelFreq,xaxis.labelFormat	see Details.
epochsAsDate	Boolean indicating whether to treat the epochs as Date objects. Default: Value of the corresponding slot in x.

`xlab` a title for the x axis. See `plot.default`.

`ylab` a title for the y axis. See `plot.default`.

`main` an overall title for the plot: see 'title'.

`type` type of plot to do.

`lty` vector of length 3 specifying the line type for the three lines in the plot – see `col` argument.

`col` Vector of length 3 specifying the color to use in the plot. The first color is the fill color of the polygons for the counts bars (`NA` for unfilled), the 2nd element denotes their border color, the 3rd element is the color of the `upperbound` plotting.

`lwd` Vector of length 3 specifying the line width of the three elements to plot. See also the `col` argment.

`alarm.symbol` a list with entries `pch`, `col`, `cex` and `lwd` specifying the appearance of the outbreak symbol in the plot.

`outbreak.symbol`
a list with entries `pch`, `col`, `cex` and `lwd` specifying the appearance of the outbreak symbol in the plot.

`legend.opts` a list containing the entries to be sent to the `legend` function. If no legend is requested use `legend.opts=NULL`. Otherwise, the following arguments are default

`x top`

`legend` The names infected and outbreak.

`lty` If `NULL` the `lty` argument will be used

`pch` If `NULL` the `pch` argument is used

`col` If `NULL` the `col` argument is used

Any further arguments to the `legend` function are just provided as additional elements of this list, e.g. `horiz=TRUE`.

`dx.upperbound` horizontal change in the plotting of the upperbound line. Sometimes it can be convenient to offset this line a little for better visiability.

`lvl` A vector of length `ncol(x)`, which is used to specify the hierarchy level for each time series in the sts object for alarm plots.

`cex` A numerical value giving the amount by which plotting text and symbols should be magnified relative to the default. See `?par` for details.

`cex.yaxis` The magnification to be used for y-axis annotation relative to the current setting of 'cex'.

`hookFunc` a function that is called after all the basic plotting has be done, i.e., it is not possible to control formatting with this function. See Examples.

`.hookFuncInheritance`
a function which is altered by sub-classes plot method. Do not alter this function manually.

`...` further arguments for the function `matplot`. If e.g. `xlab` or `main` are provided they overwrite the default values.

Details

The time series plot relies on the work-horse `stsplot_time1`. Its arguments are (almost) similiar to `plot.survRes`.

In case the epochs of the `sts` object are `Date` objects it is possible to obtain very flexible formatting of the x-axis and its annotations using the `xaxis.tickFreq`, `xaxis.labelFreq` and `xaxis.labelFormat`. The first two are named lists containing pairs with the *name* being a `strftime` single conversion specification and the second part is a function which based on this conversion returns a subset of the rows in the `sts` objects. The subsetting function has the following header: `function(x,xm1)`, where `x` is a vector containing the result of applying the conversion in `name` to the epochs of the `sts` object and `xm1` is the scalar result when applying the conversion to the natural element just before the first epoch. Three predefined subsetting functions exist: `atChange`, `at2ndChange` and `atMedian`, which are used to make a tick at each (each 2nd for `at2ndChange`) change and at the median index computed on all having the same value, respectively:

```
 atChange <- function(x,xm1) which(diff(c(xm1,x)) != 0)
 at2ndChange <- function(x,xm1) which(diff(c(xm1,x) %/% 2) != 0)
atMedian <- function(x,xm1) tapply(seq_along(x), INDEX=x, quantile, prob=0.5, type=3)
```

By defining own functions here, one can obtain an arbitrary degree of flexibility. Finally, `xaxis.labelFormat` is a `strftime` compatible formatting string., e.g. the default value is `"%G\n\n%OQ"`, which means ISO year and quarter (in roman letters) stacked on top of each other.

Value

`NULL` (invisibly). The functions are called for their side-effects.

Author(s)

Michael Höhle and Sebastian Meyer

See Also

the central `stsplot`-documentation for an overview of plot types.

Examples

```
data("ha.sts")
print(ha.sts)

plot(ha.sts, type=observed ~ time | unit)  # default multivariate type
plot(ha.sts, type=observed ~ time)         # aggregated over all districts

## Hook function example
hookFunc <- function() grid(NA,NULL,lwd=1)
plot(ha.sts, hookFunc=hookFunc)

## Use ISO8601 date formatting (see ?strptime) and no legend
data("salmNewport")
plot(aggregate(salmNewport,by="unit"), xlab="Time (weeks)",
```

```
     xaxis.tickFreq=list("%m"=atChange,"%G"=atChange),
     xaxis.labelFreq=list("%G"=atMedian),xaxis.labelFormat="%G",
     legend.opts=NULL)

## Formatting now also works for daily data (illustrate by artifical
## outbreak converted to sts object by linelist2sts)
set.seed(123)
exposureTimes <-  as.Date("2014-03-12") + sample(x=0:25,size=99,replace=TRUE)
sts <- linelist2sts(data.frame(exposure=exposureTimes),
                                dateCol="exposure",aggregate.by="1 day")
## Plot it with larger ticks for days than usual
surveillance.options("stsTickFactors"=c("%d"=1, "%W"=0.33,
                "%V"=0.33, "%m"=1.75, "%Q"=1.25, "%Y"=1.5, "%G"=1.5))
plot(sts,xaxis.tickFreq=list("%d"=atChange,"%m"=atChange),
     xaxis.labelFreq=list("%d"=at2ndChange),xaxis.labelFormat="%d-%b",
     legend.opts=NULL, xlab="Time (days)")
```

`stsSlot-generics` *Generic functions to access* `"sts"` *slots*

Description

For almost every slot of the `"sts"` class, package **surveillance** defines a generic function of the same name (except for the `population` method where the slot is actually called `populationFrac`, and `alarms`, where the slot is actually called `alarm`) as well as a replacement version (`<-`) to extract or set the corresponding slot of a `sts` object. (This documentation is not really valid yet.)

See Also

the `"sts"` class

`sts_animate` *Animated Maps and Time Series of Disease Incidence*

Description

The `animate`-method for `sts` objects supersedes the `stsplot` type `observed~1|unit*time` implemented by the function `stsplot_spacetime`. Maps generated by `stsplot_space` are sequentially plotted along time (optionally showing cumulative counts), with an optional time series chart below the map to track the epidemic curve. It is worth using functionality of the **animation** package (e.g., `saveHTML`) to directly export the animation into a useful format. See Meyer and Held (2014, Supplement A) for an example with the `fluBYBW` data.

Usage

```
## S3 method for class 'sts'
animate(object, tps = NULL, cumulative = FALSE,
        population = NULL, at = 10, ...,
        timeplot = list(height = 0.3), sleep = 0.5, verbose = interactive())
```

Arguments

object	an object of class "sts" or a matrix of counts, i.e., observed(stsObj), where especially colnames(x) have to be contained in row.names(map). If a matrix, the map object has to be provided explicitly (as part of ...).
tps	a numeric vector of one or more time points at which to plot the map. The default tps=NULL means the whole time period 1:nrow(object).
cumulative	logical specifying if the cumulative counts over time should be plotted.
population,at,...	arguments for stsplot_space.
timeplot	if a list (of arguments for the internal function stsplot_timeSimple), a time series chart of the counts along the selected time points tps will be plotted below the map. The argument height gives the relative height of the time series plot (default: 0.3), and the arguments inactive and active are lists of graphical parameters (e.g., col) determining the appearance of the bars (e.g., default color is grey when inactive and black when active).
sleep	time to wait (Sys.sleep) between subsequent snapshots (only if dev.interactive), in seconds.
verbose	logical indicating if a txtProgressBar should be shown during generation of the animation – which may take a while. Default is to do so in interactive sessions.

Value

NULL (invisibly)

Author(s)

Sebastian Meyer

References

Meyer, S. and Held, L. (2014): Power-law models for infectious disease spread. *The Annals of Applied Statistics*, **8** (3), 1612-1639.
DOI-Link: http://dx.doi.org/10.1214/14-AOAS743
Supplement A is available from http://www.biostat.uzh.ch/static/powerlaw/.

See Also

the other plot types documented in stsplot for static time series plots and maps.

Examples

```
data("measlesWeserEms")

## sequential plot of the counts by region in weeks 12-16 only (for speed)
if (require("animation")) {
    oldwd <- setwd(tempdir())  # to not clutter up the current working dir
    saveHTML(animate(measlesWeserEms, tps=12:16, cumulative=FALSE),
             title="Evolution of the measles epidemic in the Weser-Ems region",
             ani.width=500, ani.height=600)
    setwd(oldwd)
}
```

`surveillance.options` *Options of the* **surveillance** *Package*

Description

Query, set or reset options specific to the **surveillance** package, similar to what `options` does for global settings.

Usage

```
surveillance.options(...)
reset.surveillance.options()
```

Arguments

`...` Either empty, or a sequence of option names (as strings), or a sequence of `name=value` pairs, or a named list of options. Available options are:

gpclib: Logical flag indicating whether **gpclib**, the General Polygon Clipping Library for R, which has a restricted license (commercial use prohibited), may be used. This is no longer required since package **surveillance** has switched to alternatives such as **polyclip** and **rgeos** for generating `"epidataCS"` objects by `as.epidataCS` or `simEpidataCS`. However, for `unionSpatialPolygons` and `intersectPolyCircle.gpc.poly`, using **gpclib** is still an option (mainly for backwards compatibility). The default setting is `FALSE`.

stsTickFactors: A named list containing tick sizes of sts xaxis relative to par()$tcl. Each entry contains the size at `strptime` formatting strings. See the help of the plotting function for further details.

"%d"

"%W"

"%V"

"%m"

"%Q"

"%Y"

"%G"

colors: A named list containing plotting color defaults.

nowSymbol Color of the "now" symbol in `stsNC` plots. Default: `"springgreen4"`.

piBars Color of the prediction interval bars in `stsNC` plots. Default: `"orange"`.

allExamples: Logical flag mainly for CRAN-compatibility, i.e. to prevent cumbersome computations in help file examples from being run by CRAN servers. This option defaults to `TRUE` unless the environment variable `_R_CHECK_TIMINGS_` is set when attaching the **surveillance** package.

Value

`reset.surveillance.options` reverts all options to their default values and (invisibly) returns these in a list.

For `surveillance.options`, the following holds:

- If no arguments are given, the current values of all package options are returned in a list.
- If one option name is given, the current value of this option is returned (*not* in a list, just the value).
- If several option names are given, the current values of these options are returned in a list.
- If `name=value` pairs are given, the named options are set to the given values, and the *previous* values of these options are returned in a list.

Author(s)

Sebastian Meyer, inspired by the implementation of `spatstat.options()` in the **spatstat** package by Adrian Baddeley and Rolf Turner.

Examples

```
surveillance.options()
```

---

test *Print xtable for several diseases and the summary*

Description

Just a test method

Usage

```
test(data = c("k1", "m5"), range = 157:339)
```

Arguments

`data`	vector of abbreviations for the diseases
`range`	timepoints to evaluate

Details

The specified datasets are readed, corrected, enlarged and sent to the RKI 1, RKI 2, RKI 3 and Bayes system. The quality values are computed and printed for each diesease as latex table. Additonally a summary latex table for all diseases is printed

Value

xtable	printed latex tables

Author(s)

M. Höhle, A. Riebler, C. Lang

Examples

```
test(
   c("m1", "m2", "m3", "m4", "m5", "q1_nrwh", "q2",
     "s1", "s2", "s3", "k1", "n1", "n2", "h1_nrwrp")
)
```

---

testSim *Print xtable for a Simulated Disease and the Summary*

Description

Just a test method.

Usage

```
testSim(p = 0.99, r = 0.01, length = 400, A = 1, alpha = 1,
        beta = 0, phi = 0, frequency = 1, state = NULL, K,
        range = 200:400)
```

Arguments

p	probability to get a new epidemy at time i if there was one at time i-1, default 0.99
r	probability to get no new epidemy at time i if there was none at time i-1, default 0.01
length	number of weeks to model, default 400
A	amplitude (range of sinus), default = 1
alpha	parameter to move along the y-axis (negative values not allowed) with alpha >= A, default = 1
beta	regression coefficient, default = 0
phi	factor to create seasonal moves (moves the curve along the x-axis), default = 0

frequency	factor to determine the oscillation-frequency, default = 1
state	use a state chain to define the status at this timepoint (outbreak or not). If not given a Markov chain is generated by the programme, default NULL
K	additional weigth for an outbreak which influences the distribution parameter mu, default = 0
range	range of timepoints to be evaluated by the RKI 1 system, default 200:400.

Details

A pointSource epidemy is generated and sent to the RKI 1 system, the quality values for the result are computed and shown as a latex table. Additionally a plot of the result is generated.

Value

xtable	one printed latex table and a result plot

Author(s)

M. Höhle, A. Riebler, C. Lang

See Also

sim.pointSource, algo.call, algo.compare, plot.survRes, compMatrix.writeTable

Examples

```
testSim(K = 2)
testSim(r = 0.5, K = 5)
```

---

toFileDisProg *Writing of Disease Data*

Description

Writing of disease data (disProg object) into a file.

Usage

```
toFileDisProg(disProgObj, toFile)
```

Arguments

disProgObj	The disProgObj to save in file
toFile	The path and filename of the file to save

Details

Writing of disProg object into a file as illustrated in the example.

Value

file	The file with the disease data

See Also

readData, sim.pointSource

Examples

```
disProgObj <- sim.pointSource(length=200, K=1)
toFileDisProg(disProgObj, "./simulation.txt")
mydisProgObj <- readData("./simulation",sysPath=FALSE)
```

---

toLatex.sts toLatex-*Method for (lists of)* "sts" *Objects*

Description

Convert "sts" objects to a character vector with LaTeX markup.

Usage

```
## S4 method for signature 'sts'
toLatex(object, caption = "",label=" ", columnLabels = NULL,
                        subset = NULL,
                        alarmPrefix = "\\textbf{\\textcolor{red}{",
                        alarmSuffix = "}}", ubColumnLabel = "UB", ...)
```

Arguments

object	an "sts" object.
caption	A caption for the table. Default is the empty string.
label	A label for the table. Default is the empty string.
columnLabels	A list of labels for each column of the resulting table. Default is NULL
subset	A range of values which should be displayed. If Null, then all data in the sts objects will be displayed. Else only a subset of data. Therefore range needs to be a numerical vector of indexes from 1 to length(@observed).
alarmPrefix	A latex compatible prefix string wrapped around a table cell iff there is an alarm;i.e. alarm = TRUE
alarmSuffix	A latex compatible suffix string wrapped around a table cell iff there is an alarm;i.e. alarm[i,j] = TRUE
ubColumnLabel	The label of the upper bound column; default is \"UB\".
...	further arguments passed to print.xtable.

Value

An object of class "Latex".

Author(s)

Dirk Schumacher

Examples

```
# Create a test object
data("salmonella.agona")

# Create the corresponding sts object from the old disProg object
salm <- disProg2sts(salmonella.agona)

control <- list(range=(260:312),
                noPeriods=1,populationOffset=FALSE,
                fitFun="algo.farrington.fitGLM.flexible",
                b=4,w=3,weightsThreshold=1,
                pastWeeksNotIncluded=3,
                pThresholdTrend=0.05,trend=TRUE,
                thresholdMethod="delta",alpha=0.1)
salm <- farringtonFlexible(salm,control=control)
print(toLatex(salm))
```

---

twinSIR *Fit an Additive-Multiplicative Intensity Model for SIR Data*

Description

twinSIR is used to fit additive-multiplicative intensity models for epidemics as described in Höhle (2009). Estimation is driven by (penalized) maximum likelihood in the point process frame work. Optimization (maximization) of the (penalized) likelihood function is performed by means of optim.

Usage

```
twinSIR(formula, data, weights, subset,
        knots = NULL, nIntervals = 1, lambda.smooth = 0, penalty = 1,
        optim.args = list(), model = TRUE, keep.data = FALSE)
```

Arguments

formula	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the intensity model to be estimated. The details of model specification are given under Details.
data	an object inheriting from class "epidata".

`weights` an optional vector of weights to be used in the fitting process. Should be `NULL` (the default, i.e. all observations have unit weight) or a numeric vector.

`subset` an optional vector specifying a subset of observations to be used in the fitting process. The subset `atRiskY == 1` is automatically chosen, because the likelihood only depends on those observations.

`knots` numeric vector or `NULL` (the default). Specification of the knots, where we suppose a step of the log-baseline. With the current implementation, these must be existing `"stop"` time points in `subset(data, atRiskY == 1)`. The intervals of constant log-baseline hazard rate then are $(minTime; knots_1]$, $(knots_1; knots_2]$, $\ldots$, $(knots_K; maxTime]$. By default, the `knots` are automatically chosen at the quantiles of the infection time points such that `nIntervals` intervals result. Non-NULL `knots` take precedence over `nIntervals`.

`nIntervals` the number of intervals of constant log-baseline hazard. Defaults to 1, which means an overall constant log-baseline hazard will be fitted.

`lambda.smooth` numeric, the smoothing parameter λ. By default it is 0 which leads to unpenalized likelihood inference. In case `lambda.smooth=-1`, the automatic smoothing parameter selection based on a mixed model approach is used (cf. Höhle, 2009).

`penalty` either a single number denoting the order of the difference used to penalize the log-baseline coefficients (defaults to 1), or a more specific penalty matrix K for the parameter sub-vector β. In case of non-equidistant knots – usually the case when using quantile based knot locations – a 1st order differences penalty matrix as in Fahrmeir and Lang (2001) is available. For non-equidistant knots higher orders than one are not implemented.

`optim.args` a list with arguments passed to the `optim` function. Especially useful are the following ones:

- `par`: to specify initial parameter values. Those must be in the order `c(alpha, h0, beta)`, i.e. first the coefficients of the epidemic covariates in the same order as they appear in the `formula`, then the log-baseline levels in chronological order and finally the coefficients of the endemic covariates in the same order as they appear in the `cox` terms of the `formula`. The default is to start with 1's for `alpha` and 0's for `h0` and `beta`.
- `control`: for more detailed `trace`-ing (default: 1), another `REPORT`-ing frequency if `trace` is positive (default: 10), higher `maxit` (maximum number of iterations, default: 300) or another `factr` value (default: 1e7, a lower value means higher precision).
- `method`: the optimization algorithm defaults to `"L-BFGS-B"` (for box-constrained optimization), if there are any epidemic (non-`cox`) variables in the model, and to `"BFGS"` otherwise.
- `lower`: if `method = "L-BFGS-B"` this defines the lower bounds for the model coefficients. By default, all effects α of epidemic variables are restricted to be non-negative. Normally, this is exactly what one would like to have, but there might be reasons for other lower bounds, see the Note below.
- `hessian`: An estimation of the Expected Fisher Information matrix is always part of the return value of the function. It might be interesting to see the Observed Fisher Information (= negative Hessian at the maximum), too. This will be additionally returned if `hessian = TRUE`.

`model` logical indicating if the model frame, the `weights`, `lambda.smooth`, the penalty matrix K and the list of used distance functions `f` (from `attributes(data)`) should be returned for further computation. This defaults to `TRUE` as this information is necessary e.g. in the `profile` and `plot` methods.

`keep.data` logical indicating if the `"epidata"` object (`data`) should be part of the return value. This is only necessary for use of the `simulate`-method for `"twinSIR"` objects. The reason is that the `twinSIR` function only uses and stores the rows with `atRiskY == 1` in the `model` component, but for the simulation of new epidemic data one needs the whole data set with all individuals in every time block. The default value is `FALSE`, so if you intent to use `simulate.twinSIR`, you have to set this to `TRUE`.

Details

A model is specified through the `formula`, which has the form

```
~ epidemicTerm1 + epidemicTerm2 + cox(endemicVar1) *     cox(endemicVar2),
```

i.e. the right hand side has the usual form as in `lm` with some variables marked as being endemic by the special function `cox`. The left hand side of the formula is empty and will be set internally to `cbind(start, stop, event)`, which is similar to `Surv(start, stop, event, type="counting")`.

Basically, the additive-multiplicative model for the infection intensity $\lambda_i(t)$ for individual i is

$$\lambda_i(t) = Y_i(t) * (e_i(t) + h_i(t))$$

where

Y_i(t) is the at-risk indicator, indicating if individual i is "at risk" of becoming infected at time point t. This variable is part of the event history `data`.

e_i(t) is the epidemic component of the infection intensity, defined as

$$e_i(t) = \sum_{j \in I(t)} f(||s_i - s_j||)$$

where $I(t)$ is the set of infectious individuals just before time point t, s_i is the coordinate vector of individual i and the function f is defined as

$$f(u) = \sum_{m=1}^{p} \alpha_m B_m(u)$$

with unknown transmission parameters α and known distance functions B_m. This set of distance functions results in the set of epidemic variables normally calculated by the converter function `as.epidata`, considering the equality

$$e_i(t) = \sum_{m=1}^{p} \alpha_m x_{im}(t)$$

with $x_{im}(t) = \sum_{j \in I(t)} B_m(||s_i - s_j||)$ being the m'th epidemic variable for individual i.

h_i(t) is the endemic (cox) component of the infection intensity, defined as

$$h_i(t) = \exp(h_0(t) + z_i(t)'\beta)$$

where $h_0(t)$ is the log-baseline hazard function, $z_i(t)$ is the vector of endemic covariates of individual i and β is the vector of unknown coefficients. To fit the model, the log-baseline hazard function is approximated by a piecewise constant function with known knots, but unknown levels, which will be estimated. The approximation is specified by the arguments `knots` or `nIntervals`.

If a big number of `knots` (or `nIntervals`) is chosen, the corresponding log-baseline parameters can be rendered identifiable by the use of penalized likelihood inference. At present, it is the job of the user to choose an adequate value of the smoothing parameter `lambda.smooth`. Alternatively, a data driven `lambda.smooth` smoothing parameter selection based on a mixed model representation of an equivalent truncated power spline is offered (see reference for further details). The following two steps are iterated until converegence:

1. Given fixed smoothing parameter, the penalized likelihood is optimized for the regression components using a L-BFGS-B approach
2. Given fixed regression parameters, a Laplace approximation of the marginal likelihood for the smoothing parameter is numerically optimized.

Depending on the data convergence might take a couple of iterations.

Note also that it is unwise to include endemic covariates with huge values, as they affect the intensities on the exponential scale after having been multiplied by the parameter vector β. With big covariates the `optim` method "L-BFGS-B" will likely terminate due to an infinite log-likelihood or score function in some iteration.

Value

`twinSIR` returns an object of `class` `"twinSIR"`. An object of this class is a list containing the following components:

`coefficients`	a named vector of coefficients.
`loglik`	the maximum of the (penalized) log-likelihood function.
`counts`	the number of log-likelihood and score function evaluations.
`converged`	logical indicating convergence of the optimization algorithm.
`fisherinfo.observed`	if requested, the negative Hessian from `optim`.
`fisherinfo`	an estimation of the Expected Fisher Information matrix.
`method`	the optimization algorithm used.
`intervals`	a numeric vector (`c(minTime, knots, maxTime)`) representing the consecutive intervals of constant log-baseline.
`nEvents`	a numeric vector containing the number of infections in each of the above `intervals`.
`model`	if requested, the model information used. This is a list with components `"survs"` (data.frame with the id, start, stop and event columns), `"X"` (matrix of the epidemic variables), `"Z"` (matrix of the endemic variables), `"weights"` (the specified `weights`), `"lambda.smooth"` (the specified `lambda.smooth`), `"K"` (the penalty

	matrix used), and "f" and "w" (the functions to generate the used epidemic covariates). Be aware that the model only contains those rows with atRiskY == 1!
data	if requested, the supplied "epidata" data.
call	the matched call.
formula	the specified formula.
terms	the terms object used.

Note

There are some restrictions to modelling the infection intensity without a baseline hazard rate, i.e. without an intercept in the formula. Reason: At some point, the optimization algorithm L-BFGS-B tries to set all transmission parameters α to the boundary value 0 and to calculate the (penalized) score function with this set of parameters (all 0). The problem then is that the values of the infection intensities $lambda_i(t)$ are 0 for all i and t and especially at observed event times, which is impossible. Without a baseline, it is not allowed to have all alpha's set to 0, because then we would not observe any infections. Unfortunately, L-BFGS-B can not consider this restriction. Thus, if one wants to fit a model without baseline hazard, the control parameter lower must be specified in optim.args so that some alpha is strictly positive, e.g. optim.args = list(lower = c(0,0.001,0.001,0)) and the initial parameter vector par must not be the zero vector.

Author(s)

Michael Höhle and Sebastian Meyer

References

Höhle, M. (2009), Additive-Multiplicative Regression Models for Spatio-Temporal Epidemics, Biometrical Journal, 51(6):961-978.

See Also

as.epidata for the necessary data input structure, plot.twinSIR for plotting the path of the infection intensity, profile.twinSIR for profile likelihood estimation. and simulate.twinSIR for the simulation of epidemics following the fitted model.

Furthermore, the standard extraction methods vcov, logLik, AIC and extractAIC are implemented for objects of class "twinSIR".

Examples

```
data("fooepidata")
summary(fooepidata)

# fit an overall constant baseline hazard rate
fit1 <- twinSIR(~ B1 + B2 + cox(z2), data = fooepidata)
fit1
summary(fit1)

# fit1 is what is used as data("foofit") in other examples
data("foofit")
```

```
stopifnot(all.equal(fit1, foofit))

# fit a piecewise constant baseline hazard rate with 3 intervals using
# _un_penalized ML and estimated coefs from fit1 as starting values
fit2 <- twinSIR(~ B1 + B2 + cox(z2), data = fooepidata, nIntervals = 3,
    optim.args = list(par=c(coef(fit1)[1:2],rep(coef(fit1)[3],3),coef(fit1)[4])))
fit2
summary(fit2)

# fit a piecewise constant baseline hazard rate with 9 intervals
# using _penalized_ ML and estimated coefs from fit1 as starting values
fit3 <- twinSIR(~ B1 + B2 + cox(z2), data = fooepidata, nIntervals = 9,
    lambda.smooth = 0.1, penalty = 1, optim.args = list(
        par=c(coef(fit1)[1:2], rep(coef(fit1)[3],9), coef(fit1)[4])))
fit3
summary(fit3)
# plot of the 9 log-baseline levels
plot(x=fit3$intervals, y=coef(fit3)[c(3,3:11)], type="S")

### -> for more sophisticated intensity plots, see 'plot.twinSIR'
plot(fit3)
```

`twinSIR_intensityplot` *Plotting Paths of Infection Intensities for* `twinSIR` *Models*

Description

`intensityplot` methods to plot the evolution of the total infection intensity, its epidemic proportion or its endemic proportion over time. The default `plot` method for objects of class `"twinSIR"` is just a wrapper for the `intensityplot` method.

Usage

```
## S3 method for class 'twinSIR'
plot(x, which = c("epidemic proportion", "endemic proportion",
     "total intensity"), ...)

## S3 method for class 'twinSIR'
intensityplot(x, which = c("epidemic proportion", "endemic proportion",
              "total intensity"), aggregate = TRUE, theta = NULL,
              plot = TRUE, add = FALSE, rug.opts = list(), ...)

## S3 method for class 'simEpidata'
intensityplot(x, which = c("epidemic proportion", "endemic proportion",
              "total intensity"), aggregate = TRUE, theta = NULL,
              plot = TRUE, add = FALSE, rug.opts = list(), ...)
```

Arguments

x	an object of class "twinSIR" (fitted model) or "simEpidata" (simulated twinSIR epidemic), respectively.
which	"epidemic proportion", "endemic proportion", or "total intensity". Partial matching is applied. Determines whether to plot the path of the total intensity $\lambda(t)$ or its epidemic or endemic proportions $\frac{e(t)}{\lambda(t)}$ or $\frac{h(t)}{\lambda(t)}$.
aggregate	logical. Determines whether lines for all individual infection intensities should be drawn (FALSE) or their sum only (TRUE, the default).
theta	numeric vector of model coefficients. If x is of class "twinSIR", then theta = c(alpha, beta), where beta consists of the coefficients of the piecewise constant log-baseline function and the coefficients of the endemic (cox) predictor. If x is of class "simEpidata", then theta = c(alpha, 1, betarest), where 1 refers to the (true) log-baseline used in the simulation and betarest is the vector of the remaining coefficients of the endemic (cox) predictor. The default (NULL) means that the fitted or true parameters, respectively, will be used.
plot	logical indicating if a plot is desired, defaults to TRUE. Otherwise, only the data of the plot will be returned. Especially with aggregate = FALSE and many individuals one might e.g. consider to plot a subset of the individual intensity paths only or do some further calculations/analysis of the infection intensities.
add	logical. If TRUE, paths are added to the current plot, using lines.
rug.opts	either a list of arguments passed to the function rug, or NULL (or NA), in which case no rug will be plotted. By default, the argument ticksize is set to 0.02 and quiet is set to TRUE. Note that the argument x of the rug() function, which contains the locations for the rug is fixed internally and can not be modified. The locations of the rug are the time points of infections.
...	For the plot.twinSIR methode, arguments passed to intensityplot.twinSIR. For the intensityplot methods, further graphical parameters passed to the function matplot, e.g. lty, lwd, col, xlab, ylab and main. Note that the matplot arguments x, y, type and add are implicit and can not be specified here.

Value

numeric matrix with the first column "stop" and as many rows as there are "stop" time points in the event history x. The other columns depend on the argument aggregate: if TRUE, there is only one other column named which, which contains the values of which at the respective "stop" time points. Otherwise, if aggregate = FALSE, there is one column for each individual, each of them containing the individual which at the respective "stop" time points.

Author(s)

Sebastian Meyer

References

Höhle, M. (2009), Additive-Multiplicative Regression Models for Spatio-Temporal Epidemics, Biometrical Journal, 51(6):961-978.

See Also

twinSIR or Höhle (2009) for a more detailed description of the intensity model. simulate.twinSIR for the simulation of epidemic data according to a twinSIR specification.

Examples

```
data("fooepidata")
data("foofit")

# an overview of the evolution of the epidemic
plot(fooepidata)

# overall total intensity
plot(foofit, which="total")

# overall epidemic proportion
epi <- plot(foofit, which="epidemic")

# look at returned values
head(epi)

# add the inverse overall endemic proportion = 1 - epidemic proportion
ende <- plot(foofit, which="endemic", add=TRUE, col=2)
legend("right", legend="endemic proportion \n(= 1 - epidemic proportion)",
       lty=1, col=2, bty="n")

# individual intensities
tmp <- plot(foofit, which="total", aggregate=FALSE, col=rgb(0,0,0,alpha=0.1),
    main=expression("Individual infection intensities " *
                    lambda[i](t) == Y[i](t) %.% (e[i](t) + h[i](t))))
# return value: matrix of individual intensity paths
str(tmp)

# plot intensity path only for individuals 3 and 99
matplot(x=tmp[,1], y=tmp[,1+c(3,99)], type="S", ylab="Force of infection",
        xlab="time", main=expression("Paths of the infection intensities " *
                                     lambda[3](t) * " and " * lambda[99](t)))
legend("topright", legend=paste("Individual", c(3,99)), col=c(1,2), lty=c(1,2))
```

---

twinSIR_methods *Print, Summary and Extraction Methods for "twinSIR" Objects*

Description

Besides print and summary methods there are also some standard extraction methods defined for objects of class "twinSIR": vcov, logLik and especially AIC and extractAIC, which extract Akaike's Information Criterion. Note that special care is needed, when fitting models with parameter constraints such as the epidemic effects α in twinSIR models. Parameter constraints reduce the average increase in the maximized loglikelihood - thus the penalty for constrained parameters

should be smaller than the factor 2 used in the ordinary definition of AIC. To this end, these two methods offer the calculation of the so-called one-sided AIC (OSAIC).

Usage

```
## S3 method for class 'twinSIR'
print(x, digits = max(3, getOption("digits") - 3), ...)
## S3 method for class 'twinSIR'
summary(object,
        correlation = FALSE, symbolic.cor = FALSE, ...)

## S3 method for class 'twinSIR'
AIC(object, ..., k = 2, one.sided = NULL, nsim = 1e3)
## S3 method for class 'twinSIR'
extractAIC(fit, scale = 0, k = 2, one.sided = NULL,
           nsim = 1e3, ...)

## S3 method for class 'twinSIR'
vcov(object, ...)
## S3 method for class 'twinSIR'
logLik(object, ...)

## S3 method for class 'summary.twinSIR'
print(x,
      digits = max(3, getOption("digits") - 3), symbolic.cor = x$symbolic.cor,
      signif.stars = getOption("show.signif.stars"), ...)
```

Arguments

`x, object, fit`	an object of class `"twinSIR"`. For the `print` method of the `summary` method, an object of class `"summary.twinSIR"`.
`digits`	integer, used for number formatting with `signif()`. Minimum number of significant digits to be printed in values.
`correlation`	logical. if `TRUE`, the correlation matrix of the estimated parameters is returned and printed.
`symbolic.cor`	logical. If `TRUE`, print the correlations in a symbolic form (see `symnum`) rather than as numbers.
`...`	For the `summary` method: arguments passed to `extractAIC.twinSIR`. For the `AIC` method, optionally more fitted model objects. For the `print`, `extractAIC`, `vcov` and `logLik` methods: unused (argument of the generic).
`k`	numeric specifying the "weight" of the *penalty* to be used; in an unconstrained fit `k = 2` is the classical AIC.
`one.sided`	logical or `NULL` (the default). Determines if the one-sided AIC should be calculated instead of using the classical penalty `k*edf`. The default value `NULL` chooses classical AIC in the case of an unconstrained fit and one-sided AIC in the case of constraints. The type of the fit can be seen in `object$method` (or `fit$method` respectively), where `"L-BFGS"` means constrained optimization.

`nsim`	when there are more than two epidemic covariates in the fit, the weights in the OSAIC formula have to be deterimed by simulation. Default is to use 1000 samples. Note that package **quadprog** is additionally required in this case.
`scale`	unused (argument of the generic).
`signif.stars`	logical. If `TRUE`, "significance stars" are printed for each coefficient.

Details

The `print` and `summary` methods allow the compact or comprehensive representation of the fitting results, respectively. The former only prints the original function call, the estimated coefficients and the maximum log-likelihood value. The latter prints the whole coefficient matrix with standard errors, z- and p-values (see `printCoefmat`), and additionally the number of infections per log-baseline `interval`, the (one-sided) AIC and the number of log-likelihood evaluations. They both append a big "WARNING", if the optimization algorithm did not converge.

The estimated coefficients may be extracted by using the default `coef`-method from package **stats**.

The two AIC functions differ only in that `AIC` can take more than one fitted model object and that `extractAIC` always returns the number of parameters in the model (`AIC` only does with more than one fitted model object).

Concerning the choice of one-sided AIC: parameter constraints – such as the non-negative constraints for the epidemic effects alpha in `twinSIR` models – reduce the average increase in the maximized loglikelihood. Thus, the penalty for constrained parameters should be smaller than the factor 2 used in the ordinary definition of AIC. One-sided AIC (OSAIC) suggested by Hughes and King (2003) is such a proposal when p out of $k = p + q$ parameters have non-negative constraints:

$$OSAIC = -2l(\theta, \tau) + 2\sum_{g=0}^{p} w(p, g)(k - p + g)$$

where $w(p, g)$ are p-specific weights. For more details see Section 5.2 in Höhle (2009).

Value

The `print` methods return their first argument, invisibly, as they always should. The `vcov` and `logLik` methods return the estimated variance-covariance matrix of the parameters (here, the inverse of the estimate of the expected Fisher information matrix), and the maximum log-likelihood value of the model, respectively. The `summary` method returns a list containing some summary statistics of the fitted model, which is nicely printed by the corresponding `print` method. For the `AIC` and `extractAIC` methods, see the documentation of the corresponding generic functions.

Author(s)

Michael Höhle and Sebastian Meyer

References

Hughes A, King M (2003) Model selection using AIC in the presence of one-sided information. *Journal of Statistical Planning and Inference* **115**, pp. 397–411.

Höhle, M. (2009), Additive-Multiplicative Regression Models for Spatio-Temporal Epidemics, Biometrical Journal, 51(6):961-978.

Examples

```
data("foofit")

foofit

coef(foofit)
vcov(foofit)
logLik(foofit)

summary(foofit, correlation = TRUE, symbolic.cor = TRUE)

# AIC or OSAIC
AIC(foofit)
AIC(foofit, one.sided = FALSE)
extractAIC(foofit)
extractAIC(foofit, one.sided = FALSE)

# just as a stupid example for the use of AIC with multiple fits
foofit2 <- foofit
AIC(foofit, foofit2)   # 2nd column should actually be named "OSAIC" here
```

---

twinSIR_profile *Profile Likelihood Computation and Confidence Intervals*

Description

Function to compute estimated and profile likelihood based confidence intervals. Computations might be cumbersome!

Usage

```
## S3 method for class 'twinSIR'
profile(fitted, profile, alpha = 0.05,
        control = list(fnscale = -1, factr = 10, maxit = 100), ...)
```

Arguments

`fitted`	an object of class `"twinSIR"`.
`profile`	a list with elements being numeric vectors of length 4. These vectors must have the form `c(index, lower, upper, gridsize)`.
	`index`**:** index of the parameter to be profiled in the vector `coef(fitted)`.
	`lower, upper`**:** lower/upper limit of the grid on which the profile log-likelihood is evaluated. Can also be `NA` in which case `lower/upper` equals the lower/upper bound of the respective 0.3 % Wald confidence interval (+-3*se).
	`gridsize`**:** grid size of the equally spaced grid between lower and upper. Can also be 0 in which case the profile log-likelihood for this parameter is not evaluated on a grid.

alpha	$(1-\alpha)100\%$ profile likelihood based confidence intervals are computed. If `alpha <= 0`, then no confidence intervals are computed.
control	control object to use in `optim` for the profile log-likelihood computations.
...	unused (argument of the generic).

Value

list with profile log-likelihood evaluations on the grid and highest likelihood and wald confidence intervals. The argument `profile` is also returned.

Author(s)

Michael Höhle and Sebastian Meyer

Examples

```
if (surveillance.options("allExamples")) {
  data("foofit")
  prof <- profile(foofit, list(c(1,NA,NA,5), c(3,NA,NA,0), c(4, 0.5, 1.1, 10)))
  prof

  ## there is also a plot-method for "profile.twinSIR"
  plot(prof)
}
```

---

twinSIR_simulation *Simulation of Epidemic Data*

Description

This function simulates the infection (and removal) times of an epidemic. Besides the classical SIR type of epidemic, also SI, SIRS and SIS epidemics are supported. Simulation works via the conditional intensity of infection of an individual, given some (time varying) endemic covariates and/or some distance functions (epidemic components) as well as the fixed positions of the individuals. The lengths of the infectious and removed periods are generated following a pre-specified function (can be deterministic).

The `simulate` method for objects of class `"twinSIR"` simulates new epidemic data using the model and the parameter estimates of the fitted object.

Usage

```
simEpidata(formula, data, id.col, I0.col, coords.cols, subset,
           beta, h0, f = list(), w = list(), alpha, infPeriod,
           remPeriod = function(ids) rep(Inf, length(ids)),
           end = Inf, trace = FALSE, .allocate = NULL)

## S3 method for class 'twinSIR'
```

```
simulate(object, nsim = 1, seed = 1,
         infPeriod = NULL, remPeriod = NULL,
         end = diff(range(object$intervals)), trace = FALSE, .allocate = NULL,
         data = object$data, ...)
```

Arguments

formula	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the intensity model to be estimated. The details of model specification are given under Details.
data	a data.frame containing the variables in formula and the variables specified by id.col, I0.col and coords.col (see below). It represents the "history" of the endemic covariates to use for the simulation. The form is similar to and can be an object of class "epidata". The simulation period is splitted up into *consecutive* intervals of constant endemic covariables. The data frame consists of a block of N (number of individuals) rows for each of those time intervals (all rows in a block share the same start and stop values... therefore the name "block"), where there is one row per individual in the block. Each row describes the (fixed) state of the endemic covariates of the individual during the time interval given by the start and stop columns (specified through the lhs of formula). For the simulate method of class "twinSIR" this should be the object of class "epidata" used for the fit. This is a part of the return value of the function twinSIR, if called with argument keep.data set to TRUE.
id.col	only if data does not inherit from epidata: single index of the id column in data. Can be numeric (by column number) or character (by column name). The id column identifies the individuals in the data-frame. It will be converted to a factor variable and its levels serve also to identify individuals as argument to the infPeriod function.
I0.col	only if data does not inherit from epidata: single index of the I0 column in data. Can be numeric (by column number), character (by column name) or NULL. The I0 column indicates if an individual is initially infectious, i.e. it is already infectious at the beginning of the first time block. Setting I0.col = NULL is short for "there are no initially infectious individuals". Otherwise, the variable must be logical or in 0/1-coding. As this variable is constant over time the initially infectious individuals are derived from the first time block only.
coords.cols	only if data does not inherit from epidata: index*es* of the coords column*s* in data. Can be a numeric (by column number), a character (by column name) vector or NULL. These columns contain the coordinates of the individuals. It must be emphasized that the functions in this package currently assume *fixed positions* of the individuals during the whole epidemic. Thus, an individual has the same coordinates in every block. For simplicity, the coordinates are derived from the first time block only. The epidemic covariates are calculated based on the Euclidian distance between the individuals, see f.
subset	an optional vector specifying a subset of the covariate history to be used in the simulation.

beta — numeric vector of length equal the number of endemic (cox) terms on the rhs of formula. It contains the effects of the endemic predictor (excluding the log-baseline h0, see below) in the same order as in the formula.

h0 — *either* a single number to specify a constant baseline hazard (equal to exp(h0)) *or* a list of functions named exact and upper. In the latter case, h0$exact is the true log-baseline hazard function and h0$upper is a *piecewise constant upper bound* for h0$exact. The function h0$upper must inherit from stepfun with right=FALSE. Theoretically, the intensity function is left-continuous, thus right=TRUE would be adequate, but in the implementation, when we evaluate the intensity at the knots (change points) of h0$upper we need its value for the subsequent interval.

f, w — see as.epidata.

alpha — a named numeric vector of coefficients for the epidemic covariates generated by f and w. The names are matched against names(f) and names(w). Remember that alpha >= 0.

infPeriod — a function generating lengths of infectious periods. It should take one parameter (e.g. ids), which is a character vector of id's of individuals, and return appropriate infection periods for those individuals. Therefore, the value of the function should be of length length(ids). For example, for independent and identically distributed infection periods following $Exp(1)$, the generating function is function(ids) rexp(length(ids), rate=1). For a constant infectious period of length c, it is sufficient to set function (x) {c}.
For the simulate method of class "twinSIR" only, this can also be NULL (the default), which means that the observed infectious periods of infected individuals are re-used when simulating a new epidemic and individuals with missing infectious periods (i.e. infection and recovery was not observed) are attributed to the mean observed infectious period.

Note that it is even possible to simulate an SI-epidemic by setting

```
infPeriod = function (x) {Inf}
```

In other words: once an individual became infected it spreads the disease forever, i.e. it will never be removed.

remPeriod — a function generating lengths of removal periods. Per default, once an individual was removed it will stay in this state forever (Inf). Therefore, it will not become at-risk (S) again and re-infections are not possible. Alternatively, always returning 0 as length of the removal period corresponds to a SIS epidemic. Any other values correspond to SIRS.

end — a single positive numeric value specifying the time point at which the simulation should be forced to end. By default, this is Inf, i.e. the simulation continues until there is no susceptible individual left.
For the simulate method of class "twinSIR" the default is to have equal simulation and observation periods.

trace — logical (or integer) indicating if (or how often) the sets of susceptible and infected individuals as well as the rejection indicator (of the rejection sampling step) should be cated. Defaults to FALSE.

.allocate — number of blocks to initially allocate for the event history (i.e. .allocate*N rows). By default (NULL), this number is set to max(500, ceiling(nBlocks/100)*100),

	i.e. 500 but at least the number of blocks in `data` (rounded to the next multiple of 100). Each time the simulated epidemic exceeds the allocated space, the event history will be enlarged by `.allocate` blocks.
`object`	an object of class `"twinSIR"`. This must contain the original `data` used for the fit (see `data`).
`nsim`	number of epidemics to simulate. Defaults to 1.
`seed`	an integer that will be used in the call to `set.seed` before simulating the epidemics.
`...`	unused (argument of the generic).

Details

A model is specified through the `formula`, which has the form

`cbind(start, stop) ~ cox(endemicVar1) * cox(endemicVar2)`,

i.e. the right hand side has the usual form as in `lm`, but all variables are marked as being endemic by the special function `cox`. The effects of those predictor terms are specified by `beta`. The left hand side of the formula denotes the start and stop columns in `data`. This can be omitted, if `data` inherits from class `"epidata"` in which case `cbind(start, stop)` will be used. The epidemic model component is specified by the arguments `f` and `w` (and the associated coefficients `alpha`).

If the epidemic model component is empty and `infPeriod` always returns `Inf`, then one actually simulates from a pure Cox model.

The simulation algorithm used is *Ogata's modified thinning*. For details, see Höhle (2009), Section 4.

Value

An object of class `"simEpidata"`, which is a `data.frame` with the columns `"id"`, `"start"`, `"stop"`, `"atRiskY"`, `"event"`, `"Revent"` and the coordinate columns (with the original names from `data`), which are all obligatory. These columns are followed by all the variables appearing on the rhs of the `formula`. Last but not least, the generated columns with epidemic covariates corresponding to the functions in the lists `f` and `w` are appended.

Note that objects of class `"simEpidata"` also inherit from class `"epidata"`, thus all `"epidata"` methods can be applied.

The `data.frame` is given the additional *attributes*

`"eventTimes"`	numeric vector of infection time points (sorted chronologically).
`"timeRange"`	numeric vector of length 2: `c(min(start), max(stop))`.
`"coords.cols"`	numeric vector containing the column indices of the coordinate columns in the resulting data-frame.
`"f"`	this equals the argument `f`.
`"w"`	this equals the argument `w`.
`"config"`	a list with elements `h0 = h0$exact`, `beta` and `alpha`.
`call`	the matched call.
`terms`	the `terms` object used.

If `nsim > 1` epidemics are simulated by the `simulate`-method for fitted `"twinSIR"` models, these are returned in a list.

Author(s)

Sebastian Meyer and Michael Höhle

References

Höhle, M. (2009), Additive-Multiplicative Regression Models for Spatio-Temporal Epidemics, Biometrical Journal, 51(6):961-978.

See Also

The `plot.epidata` and `animate.epidata` methods for plotting and animating (simulated) epidemic data, respectively. The `intensityplot.simEpidata` method for plotting paths of infection intensities.

Function `twinSIR` for fitting spatio-temporal epidemic intensity models to epidemic data.

Examples

```
## Generate a data frame containing a hypothetic population with 100 individuals
set.seed(1234)
n <- 100
pos <- matrix(rnorm(n*2), ncol=2, dimnames=list(NULL, c("x", "y")))
pop <- data.frame(id=1:n,x=pos[,1], y=pos[,2],
                  gender=sample(0:1, n, replace=TRUE),
                  I0col=rep(0,n),start=rep(0,n),stop=rep(Inf,n))

## Simulate an epidemic in this population
set.seed(1)
epi <- simEpidata(cbind(start,stop) ~ cox(gender),
                  data = pop,
                  id = "id", I0.col = "I0col", coords.cols = c("x","y"),
                  beta = c(-2), h0 = -1, alpha = c(B1 = 0.1),
                  f = list(B1 = function(u) u <= 1),
                  infPeriod = function(ids) rexp(length(ids), rate=1))

# Plot the numbers of susceptible, infectious and removed individuals
plot(epi)


## load data of an artificial epidemic
data("fooepidata")
summary(fooepidata)
plot(fooepidata)

if (surveillance.options("allExamples"))
{
  ## simulate a new evolution of the epidemic
  set.seed(1)
  simepi <- simEpidata(cbind(start, stop) ~ cox(z1) + cox(z1):cox(z2),
      data = fooepidata,
      beta = c(1,0.5), h0 = -7, alpha = c(B2 = 0.01, B1 = 0.005),
      f = list(B1 = function(u) u <= 1, B2 = function(u) u > 1),
```

```
      infPeriod = function(ids) rexp(length(ids), rate=0.2), trace = FALSE)
  summary(simepi)
  plot(simepi)
  intensityplot(simepi)
}

## load a model fitted to the 'fooepidata' epidemic
data("foofit")
foofit

## simulate a new epidemic using the model and parameter estimates of 'foofit'
## and set simulation period = observation period
# a) with observed infPeriods (i.e. fixed length 3 days):
simfitepi1 <- simulate(foofit, data=fooepidata)
plot(simfitepi1)
# b) with new infPeriods (simuluated from the Exp(0.3) distribution):
simfitepi2 <- simulate(foofit, data=fooepidata,
                       infPeriod=function(ids) rexp(length(ids), rate=0.3))
plot(simfitepi2)
intensityplot(simfitepi2, which="total", aggregate=FALSE,
              col=rgb(0,0,0,alpha=0.1))
```

`twinstim` *Fit a Two-Component Spatio-Temporal Point Process Model*

Description

A `twinstim` model as described in Meyer et al. (2012) is fitted to marked spatio-temporal point process data. This constitutes a regression approach for conditional intensity function modelling.

Usage

```
twinstim(endemic, epidemic, siaf, tiaf, qmatrix = data$qmatrix, data,
         subset, t0 = data$stgrid$start[1], T = tail(data$stgrid$stop,1),
         na.action = na.fail, start = NULL, partial = FALSE,
         control.siaf = list(F = list(), Deriv = list()),
         optim.args = list(), finetune = FALSE,
         model = FALSE, cumCIF = FALSE, cumCIF.pb = interactive(),
         cores = 1, verbose = TRUE)
```

Arguments

`endemic`	right-hand side formula for the exponential (Cox-like multiplicative) endemic component. May contain offsets (to be marked by the special function `offset`). If omitted or `~0` there will be no endemic component in the model. A type-specific endemic intercept can be requested by including the term `(1|type)` in the formula.

epidemic — formula representing the epidemic model for the event-specific covariates (marks) determining infectivity. Offsets are not implemented here. If omitted or `~0` there will be no epidemic component in the model.

siaf — spatial interaction function. Possible specifications are:

- `NULL` or missing, corresponding to `siaf.constant()`, i.e. spatially homogeneous infectivity independent of the distance from the host
- a list as returned by `siaf` or by a predefined interaction function such as `siaf.gaussian` as in Meyer et al. (2012) or `siaf.powerlaw` as in Meyer and Held (2014)
- a numeric vector corresponding to the knots of a step function, i.e. the same as `siaf.step(knots)`

If you run into "false convergence" with a non-constant `siaf` specification, the numerical accuracy of the cubature methods is most likely too low (see the `control.siaf` argument).

tiaf — temporal interaction function. Possible specifications are:

- `NULL` or missing, corresponding to `tiaf.constant()`, i.e. time-constant infectivity
- a list as returned by `tiaf` or by a predefined interaction function such as `tiaf.exponential`
- a numeric vector corresponding to the knots of a step function, i.e. the same as `tiaf.step(knots)`

qmatrix — square indicator matrix (0/1 or `FALSE`/`TRUE`) for possible transmission between the event types. The matrix will be internally converted to `logical`. Defaults to the Q matrix specified in `data`.

data — an object of class `"epidataCS"`.

subset — an optional vector evaluating to logical indicating a subset of `data$events` to keep. Missing values are taken as `FALSE`. The expression is evaluated in the context of the `data$events@data` `data.frame`, i.e. columns of this `data.frame` may be referenced directly by name.

t0, T — events having occured during (-Inf;t0] are regarded as part of the prehistory H_0 of the process. Only events that occured in the interval (t0; T] are considered in the likelihood. The time point `t0` (`T`) must be an element of `data$stgrid$start` (`data$stgrid$stop`). The default time range covers the whole spatio-temporal grid of endemic covariates.

na.action — how to deal with missing values in `data$events`? Do not use `na.pass`. Missing values in the spatio-temporal grid `data$stgrid` are not accepted.

start — a named vector of initial values for (a subset of) the parameters. The names must conform to the conventions of `twinstim` to be assigned to the correct model terms. For instance, `"h.(Intercept)"` = endemic intercept, `"h.I(start/365)"` = coefficient of a linear time trend in the endemic component, `"h.factorB"` = coefficient of the level B of the factor variable `factor` in the endemic predictor, `"e.(Intercept)"` = epidemic intercept, `"e.VAR"` = coefficient of the epidemic term `VAR`, `"e.siaf.2"` = second `siaf` parameter, `"e.tiaf.1"` = first `tiaf` parameter. Elements which don't match any of the model parameters are ignored.

Alternatively, start may also be a named list with elements "endemic" or "h", "epidemic" or "e", "siaf" or "e.siaf", and "tiaf" or "e.tiaf", each of which containing a named numeric vector with the term labels as names (i.e. without the prefix "h.", "e.", etc). Thus, start=list(endemic=c("(Intercept)"=-10)) is equivalent to start=c("h.(Intercept)"=-10).

partial — logical indicating if a partial likelihood similar to the approach by Diggle et al. (2010) should be used (default is FALSE). Note that the partial likelihood implementation is not well tested.

control.siaf — a list with elements "F" and "Deriv", which are lists of extra arguments passed to the functions siaf$F and siaf$Deriv, respectively.
These arguments mainly determine the accuracy of the numerical cubature rules from package **polyCub** involved in non-constant siaf specifications, e.g., the bandwidth of the midpoint rule polyCub.midpoint, the number of Gaussian quadrature points for polyCub.SV, or the relative tolerance of integrate in polyCub.iso.
For instance, siaf.gaussian() uses the midpoint-cubature polyCub.midpoint with an adaptive bandwidth of eps=adapt*sd to numerically integrate the kernel $f(\boldsymbol{s})$, and the default adapt value (0.1) can be overwritten by setting control.siaf$F$adapt. The default integration method for the derivative of $f(\boldsymbol{s})$ wrt the kernel parameters is product Gauss cubature polyCub.SV with nGQ=20 univariate Gauss quadrature points. This number can be overwritten by setting control.siaf$Deriv$nGQ.
This argument list is ignored in the case siaf=siaf.constant() (which is the default if siaf is unspecified).

optim.args — an argument list passed to optim, or NULL, in which case no optimization will be performed but the necessary functions will be returned in a list (similar to what is returned if model = TRUE).

Initial values for the parameters may be given as list element par in the order (endemic, epidemic, siaf, tiaf). If no initial values are provided, a crude estimate for the endemic intercept will be used (see the Examples), but just zeroes for the remaining parameters. However, any initial values given in the start argument take precedence over those in par.

Note that optim receives the negative log-likelihood for minimization (thus, if used, optim.args$control$fnscale should be positive). The hessian argument defaults to TRUE, and in the control list, tracing is enabled with REPORT=1 by default. By setting optim.args$control$trace = 0, all output from the optimization routine is suppressed.

For the partial likelihood, the analytic score function and the Fisher information are not implemented and the default is to use robust method="Nelder-Mead" optimization.

There may be an extra component fixed in the optim.args list, which determines which parameters should stick to their initial values. This can be specified by a logical vector of the same length as the par component, by an integer vector indexing par or by a character vector following the twinstim naming conventions. Furthermore, if isTRUE(fixed), then all parameters are fixed at their initial values and no optimization is performed.

Importantly, the method argument in the optim.args list may also be "nlminb", in which case the nlminb optimizer is used. This is also the default for full like-

lihood inference. In this case, not only the score function but also the *expected* Fisher information can be used during optimization (as estimated by what Martinussen and Scheike (2006, p. 64) call the "optional variation process", or see Rathbun (1996, equation (4.7))). In our experience this gives better convergence than optim's methods. For method="nlminb", the following parameters of the optim.args$control list may be named like for optim and are renamed appropriatly: maxit (-> iter.max), REPORT (-> trace, default: 1), abstol (-> abs.tol), and reltol (-> rel.tol, default: 1e-6). For nlminb, a logical hessian argument (default: TRUE) indicates if the negative *expected* Fisher information matrix should be used as the Hessian during optimization (otherwise a numerical approximation is used).

Similary, method="nlm" should also work but is not recommended here.

finetune — logical indicating if a second maximisation should be performed with robust Nelder-Mead optim using the resulting parameters from the first maximisation as starting point. This argument is only considered if partial = FALSE and the default is to not conduct a second maximization (in most cases this does not improve upon the MLE).

model — logical. If TRUE the result contains an element functions with the log-likelihood function (or partial log-likelihood function, if partial = TRUE), and optionally the score and the expected Fisher information functions (not for the partial likelihood, and only if siaf and tiaf provide the necessary derivatives). The environment of those functions equals the evaluation environment of the fitting function, i.e. it is kept in the workspace and the necessary model frames are still available when twinstim has finished. The environment is also set as an attribute of the twinstim result.

cumCIF — logical (default: FALSE) indicating whether to calculate the fitted cumulative ground intensity at event times. This is the residual process, see residuals.twinstim.

cumCIF.pb — logical indicating if a progress bar should be shown during the calculation of cumCIF. Defaults to do so in an interactive R session, and will be FALSE if cores != 1.

cores — number of processes to use in parallel operation. By default twinstim runs in single-CPU mode. Currently, only the **multicore**-type of parallel computing via forking is supported, which is not available on Windows, see mclapply in package **parallel**. Note that for a **memoise**d siaf.step kernel, cores=1 is fixed internally since parallelization would slow down model fitting significantly.

verbose — logical indicating if information should be printed during execution. Defaults to TRUE.

Details

The function performs maximum likelihood inference for the additive-multiplicative spatio-temporal intensity model described in Meyer et al. (2012). It uses nlminb as the default optimizer and returns an object of class twinstim. Such objects have print, plot and summary methods. The output of the summary can be processed by the toLatex function. Furthermore, the usual model fit methods such as coef, vcov, logLik, residuals, and update are implemented. A specific add-on is the use of the functions R0 and simulate.

Value

Returns an S3 object of class `"twinstim"`, which is a list with the following components:

`coefficients` vector containing the MLE.

`loglik` value of the log-likelihood function at the MLE with a logical attribute `"partial"` indicating if the partial likelihood was used.

`counts` number of log-likelihood and score evaluations during optimization.

`converged` either `TRUE` (if the optimizer converged) or a character string containing a failure message).

`fisherinfo` *expected* Fisher information evaluated at the MLE. Only non-`NULL` for full likelihood inference (`partial = FALSE`) and if spatial and temporal interaction functions are provided with their derivatives.

`fisherinfo.observed` observed Fisher information matrix evaluated at the value of the MLE. Obtained as the negative Hessian. Only non-`NULL` if `optim.args$method` is not `"nlminb"` and if it was requested by setting `hessian=TRUE` in `optim.args`.

`fitted` fitted values of the conditional intensity function at the events.

`fittedComponents` two-column matrix with columns `"h"` and `"e"` containing the fitted values of the endemic and epidemic components, respectively.
(Note that `rowSums(fittedComponents) == fitted`.)

`tau` fitted cumulative ground intensities at the event times. Only non-`NULL` if `cumCIF = TRUE`. This is the "residual process" of the model, see `residuals.twinstim`.

`R0` estimated basic reproduction number for each event. This equals the spatio-temporal integral of the epidemic intensity over the observation domain (t0;T] x W for each event.

`npars` vector describing the lengths of the 5 parameter subvectors: endemic intercept(s) $\beta_0(\kappa)$, endemic coefficients β, epidemic coefficients γ, parameters of the `siaf` kernel, and parameters of the `tiaf` kernel.

`qmatrix` the `qmatrix` associated with the epidemic `data` as supplied in the model call.

`bbox` the bounding box of `data$W`.

`timeRange` the time range used for fitting: `c(t0,T)`.

`formula` a list containing the four main parts of the model specification: `endemic`, `epidemic`, `siaf`, and `tiaf`.

`control.siaf` see the "Arguments" section above.

`optim.args` input optimizer arguments used to determine the MLE.

`functions` if `model=TRUE` this is a `list` with components `ll`, `sc` and `fi`, which are functions evaluating the log-likelihood, the score function and the expected Fisher information for a parameter vector θ. The `environment` of these function is the model environment, which is thus retained in the workspace if `model=TRUE`. Otherwise, the `functions` component is `NULL`.

`call` the matched call.

runtime — the proc.time-queried time taken to fit the model, i.e., a named numeric vector of length 5 of class "proc_time", with the number of cores set as additional attribute.

If model=TRUE, the model evaluation environment is assigned to this list and can thus be queried by calling environment() on the result.

Note

twinstim makes use of the **memoise** package if it is available – and that is highly recommended for non-constant siaf specifications to speed up calculations. Specifically, the necessary numerical integrations of the spatial interaction function will be cached such that they are only calculated once for every state of the siaf parameters during optimization.

Author(s)

Sebastian Meyer

Contributions to this documentation by Michael Höhle and Mayeul Kauffmann.

References

Diggle, P. J., Kaimi, I. & Abellana, R. (2010): Partial-likelihood analysis of spatio-temporal point-process data. *Biometrics*, **66**, 347-354.

Martinussen, T. and Scheike, T. H. (2006): Dynamic Regression Models for Survival Data. Springer.

Meyer, S. (2010): Spatio-Temporal Infectious Disease Epidemiology based on Point Processes. Master's Thesis, Ludwig-Maximilians-Universität München.
Available as http://epub.ub.uni-muenchen.de/11703/

Meyer, S., Elias, J. and Höhle, M. (2012): A space-time conditional intensity model for invasive meningococcal disease occurrence. *Biometrics*, **68**, 607-616.
DOI-Link: http://dx.doi.org/10.1111/j.1541-0420.2011.01684.x

Meyer, S. and Held, L. (2014): Power-law models for infectious disease spread. *The Annals of Applied Statistics*, **8** (3), 1612-1639.
DOI-Link: http://dx.doi.org/10.1214/14-AOAS743

Rathbun, S. L. (1996): Asymptotic properties of the maximum likelihood estimator for spatio-temporal point processes. *Journal of Statistical Planning and Inference*, **51**, 55-74.

See Also

twinSIR for a discrete space alternative. There is also a simulate.twinstim method, which simulates the point process based on the fitted twinstim.

Examples

```
# Load invasive meningococcal disease data
data("imdepi")

### first, fit a simple endemic-only model
```

```
## Calculate initial value for the endemic intercept (the crude estimate
## which is used by default if no initial value is supplied).
## Assume the model only had a single-cell endemic component
## (rate of homogeneous Poisson process scaled for population density):
popdensity.overall <- with(subset(imdepi$stgrid, BLOCK == 1),
    weighted.mean(popdensity, area))
popdensity.overall   # pop. density in Germany is ~230 inhabitants/km^2
W.area <- with(subset(imdepi$stgrid, BLOCK == 1), sum(area))
W.area               # area of Germany is about 357100 km^2
# this should actually be the same as
sum(sapply(imdepi$W@polygons, slot, "area"))
# which here is not exactly the case because of polygon simplification

## initial value for the endemic intercept
h.intercept <- with(summary(imdepi),
    log(nEvents/popdensity.overall/diff(timeRange)/W.area))

## fit the endemic-only model
m_noepi <- twinstim(
    endemic = addSeason2formula(~ offset(log(popdensity)) + I(start/365-3.5),
                                S=1, period=365, timevar="start"),
    data = imdepi, subset = !is.na(agegrp),
    optim.args = list(par=c(h.intercept,rep(0,3))) # the default anyway
)

## look at the model summary
summary(m_noepi)


if (surveillance.options("allExamples"))
{
  ## type-dependent endemic intercept?
  m_noepi_type <- update(m_noepi, endemic = ~(1|type) + .)
  summary(m_noepi_type)

  # LR-test for a type-dependent intercept in the endemic-only model
  pchisq(2*(logLik(m_noepi_type)-logLik(m_noepi)), df=1, lower.tail=FALSE)
}


### add an epidemic component with just the intercept, i.e.
### assuming uniform dispersal in time and space up to a distance of
### eps.s = 200 km and eps.t = 30 days (see summary(imdepi))

m0 <- update(m_noepi, epidemic=~1, start=c("e.(Intercept)"=-18), model=TRUE)

## Not run:
## NOTE: in contrast to using nlminb() optim's BFGS would miss the
##       likelihood maximum wrt the epidemic intercept if starting at -10
m0_BFGS <- update(m_noepi, epidemic=~1, start=c("e.(Intercept)"=-10),
                  optim.args = list(method="BFGS"))
format(cbind(coef(m0), coef(m0_BFGS)), digits=3, scientific=FALSE)
m0_BFGS$fisherinfo   # singular Fisher information matrix here
```

```
m0$fisherinfo
logLik(m0_BFGS)
logLik(m0)
## nlminb is more powerful since we make use of the analytical fisherinfo
## as estimated by the model during optimization, which optim cannot

## End(Not run)

## summarize the model fit
s <- summary(m0, correlation = TRUE, symbolic.cor = TRUE)
s
# output the table of coefficients as LaTeX code
toLatex(s, digits=2, withAIC=FALSE)
# or
xtable(s)

## the default confint-method can be used for Wald-CI's
confint(m0, level=0.95)

## same R0 estimate for every event (epidemic intercept only)
summary(R0(m0, trimmed=FALSE))

## plot the path of the fitted total intensity
plot(m0, "total intensity", tgrid=500)

if (surveillance.options("allExamples"))
{
  ## extract "residual process" integrating over space (takes some seconds)
  res <- residuals(m0)
  # if the model describes the true CIF well _in the temporal dimension_,
  # then this residual process should behave like a stationary Poisson
  # process with intensity 1
  plot(res, type="l"); abline(h=c(0, length(res)), lty=2)
  # -> use the function checkResidualProcess
  checkResidualProcess(m0)

  ## NB: the model fit environment is kept in the workspace with model=TRUE
  sort(sapply(environment(m0), object.size))
  # saving m0 to RData will include this model environment, thus might
  # consume quiet a lot of disk space (as it does use memory), mainly
  # depending on the size of "stgrid", the number of "events" and the
  # polygonal resolution of "W"
}

if (surveillance.options("allExamples"))
{
  ## estimate an exponential temporal decay of infectivity
  m1_tiaf <- update(m0, tiaf=tiaf.exponential())
  plot(m1_tiaf, "tiaf", scaled=FALSE)

  ### try with a spatially decreasing interaction kernel
  ## Likelihood evaluation takes much longer than for constant spatial spread
  ## Here we use the kernel of an isotropic bivariate Gaussian with same
```

```
  ## standard deviation for both finetypes

  m1 <- update(m0,
      siaf = siaf.gaussian(1, F.adaptive = TRUE),
      start = c("e.siaf.1" = 2.8),  # exp(2.8) = sigma = 16 km
      optim.args = list(fixed="e.siaf.1"),
                       # for reasons of speed of the example, the siaf
                       # parameter log(sigma) is fixed to 2.8 here
      control.siaf = list(F=list(adapt=0.25)) # use adaptive eps=sigma/4
  )

  AIC(m_noepi, m0, m1)   # further improvement
  summary(m1, test.iaf=FALSE)   # nonsense to test H0: log(sigma) = 0
  plot(m1, "siaf", scaled=FALSE)


  ### add epidemic covariates

  m2 <- update(m1, epidemic = ~ 1 + type + agegrp)

  AIC(m1, m2)   # further improvement
  summary(m2)

  # look at estimated R0 values by event type
  tapply(R0(m2), imdepi$events@data[names(R0(m2)), "type"], summary)
}
```

---

twinstim_iaf *Temporal and Spatial Interaction Functions for* twinstim

---

Description

A twinstim model as described in Meyer et al. (2012) requires the specification of the spatial and temporal interaction functions (f and g, respectively), i.e. how infectivity decays with increasing spatial and temporal distance from the source of infection. It is of course possible to define own functions (see siaf and tiaf, respectively), but the package already predefines some useful dispersal kernels returned by the constructor functions documented here.

Usage

```
# predefined spatial interaction functions
siaf.constant()
siaf.step(knots, maxRange = Inf, nTypes = 1, validpars = NULL)
siaf.gaussian(nTypes = 1, logsd = TRUE, density = FALSE,
              F.adaptive = TRUE, effRangeMult = 6, validpars = NULL)
siaf.powerlaw(nTypes = 1, validpars = NULL)
siaf.powerlawL(nTypes = 1, validpars = NULL)
siaf.student(nTypes = 1, validpars = NULL)
```

```
# predefined temporal interaction functions
tiaf.constant()
tiaf.step(knots, maxRange = Inf, nTypes = 1, validpars = NULL)
tiaf.exponential(nTypes = 1, validpars = NULL)
```

Arguments

knots	numeric vector of distances at which the step function switches to a new height. The length of this vector determines the number of parameters to estimate. For identifiability, the step function has height 1 in the first interval $[0, knots_1)$. Note that the implementation is right-continuous, i.e., intervals are $[a, b)$.
maxRange	a scalar larger than any of knots. Per default (maxRange=Inf), the step function never drops to 0 but keeps the last height for any distance larger than the last knot. However, this might not work in some cases, where the last parameter value would become very small and lead to numerical problems. It is then possible to truncate interaction at a distance maxRange (just like what the variables eps.s and eps.t do in the "epidataCS" object).
nTypes	determines the number of parameters ((log-)scales or (log-)shapes) of the kernels. In a multitype epidemic, the different types may share the same spatial interaction function, in which case nTypes=1. Otherwise nTypes should equal the number of event types of the epidemic, in which case every type has its own (log-)scale or (log-)shape, respectively. Currently, nTypes > 1 is only implemented for siaf.gaussian, tiaf.step, and tiaf.exponential.
logsd	logical indicating if the kernel should be parametrized with the log-standard deviation as the parameter in question to enforce positivity. This is the recommended default and avoids constrained optimisation (L-BFGS-B) or the use of validpars. The power-law kernels always use the log-scale for their scale and shape parameters.
density	logical indicating if the density or just its kernel should be used. If density=TRUE, siaf.gaussian uses the density of the bivariate, isotropic normal distribution as the spatial interaction function. Otherwise (default), only the kernel of the bivariate normal density is used.
F.adaptive	If F.adaptive = TRUE, then an adaptive bandwidth of adapt*sd will be used in the midpoint-cubature (polyCub.midpoint in package **polyCub**) of the Gaussian interaction kernel, where adapt is an extra parameter of the returned siaf$F function and defaults to 0.1. It can be customized either by the control.siaf$F argument list of twinstim, or by a numeric specification of F.adaptive in the constructing call, e.g., F.adaptive = 0.05 to achieve higher accuracy. Otherwise, if F.adaptive = FALSE, a general cubature method (polyCub) is returned as siaf$F component, where the method and accuracy (eps, dimyx, or nGQ, depending on the method) should then be specified in twinstim's control.siaf$F argument.
effRangeMult	determines the effective range for numerical integration in terms of multiples of the standard deviation σ of the Gaussian kernel, i.e. with effRangeMult=6 numerical integration only considers the 6σ area around the event instead of

the whole observation region `W`. Setting `effRangeMult=NULL` will disable the integral approximation with an effective integration range.

`validpars` function taking one argument, the parameter vector, indicating if it is valid (see also `siaf`). If `logsd=FALSE` and one prefers not to use `method="L-BFGS-B"` for fitting the `twinstim`, then `validpars` could be set to `function (pars) pars > 0`.

Details

The readily available spatial interaction functions are defined as follows:

`siaf.constant`: $f(s) = 1$

`siaf.step`: $f(s) = \sum_{k=0}^{K} \exp(\alpha_k) I_k(||s||)$,
where $\alpha_0 = 0$, and $\alpha_1, \ldots, \alpha_K$ are the parameters (heights) to estimate. $I_k(||s||)$ indicates if distance $||s||$ belongs to the kth interval according to `c(0,knots,maxRange)`, where $k = 0$ indicates the interval `c(0,knots[1])`.
Note that `siaf.step` makes use of the **memoise** package if it is available – and that is highly recommended to speed up calculations. Specifically, the areas of the intersection of a polygonal domain (influence region) with the "rings" of the two-dimensional step function will be cached such that they are only calculated once for every `polydomain` (in the first iteration of the `twinstim` optimization). They are used in the integration components `F` and `Deriv`. See Meyer and Held (2014) for a use case and further details.

`siaf.gaussian`: $f(s|\kappa) = \exp(-||s||/2/\sigma_\kappa^2)$
If `nTypes=1` (single-type epidemic or type-invariant `siaf` in multi-type epidemic), then $\sigma_\kappa = \sigma$ for all types κ. If `density=TRUE`, then the kernel formula above is additionally divided by $2\pi\sigma_\kappa^2$, yielding the density of the bivariate, isotropic Gaussian distribution with zero mean and covariance matrix $\sigma_\kappa^2 I_2$.

`siaf.powerlaw`: $f(s) = (||s|| + \sigma)^{-d}$,
which is the kernel of the Lomax density, i.e. without any proportionality constants. The parameters are optimized on the log-scale to ensure positivity, i.e. $\sigma = \exp(\tilde{\sigma})$ and $d = \exp(\tilde{d})$, where $(\tilde{\sigma}, \tilde{d})$ is the parameter vector.

`siaf.powerlawL`: $f(s) = (||s||/\sigma)^{-d}$, for $||s|| \geq \sigma$, and $f(s) = 1$ otherwise,
which is a *L*agged power-law kernel featuring uniform short-range dispersal (up to distance σ) and a power-law decay (Pareto-style) from distance σ onwards. The parameters are optimized on the log-scale to ensure positivity, i.e. $\sigma = \exp(\tilde{\sigma})$ and $d = \exp(\tilde{d})$, where $(\tilde{\sigma}, \tilde{d})$ is the parameter vector. However, there is a caveat associated with this kernel: Its derivative wrt $\tilde{\sigma}$ is mathematically undefined at the threshold $||s|| = \sigma$. This local non-differentiability makes `twinstim`'s likelihood maximization sensitive wrt parameter start values, and is likely to cause false convergence warnings by `nlminb`. Possible work-arounds are to use the slow and robust `method="Nelder-Mead"`, or to just ignore the warning and verify the result by sets of different start values.

`siaf.student`: $f(s) = (||s||^2 + \sigma^2)^{-d}$,
which is a reparametrized t-kernel. For $d = 1$, this is the kernel of the Cauchy density with scale `sigma`. In Geostatistics, a correlation function of this kind is known as the Cauchy model.
The parameters are optimized on the log-scale to ensure positivity, i.e. $\sigma = \exp(\tilde{\sigma})$ and $d = \exp(\tilde{d})$, where $(\tilde{\sigma}, \tilde{d})$ is the parameter vector.

The predefined temporal interaction functions are defined as follows:

`tiaf.constant`: $g(t) = 1$

`tiaf.step`: $g(t) = \sum_{k=0}^{K} \exp(\alpha_k) I_k(t)$,
where $\alpha_0 = 0$, and $\alpha_1, \ldots, \alpha_K$ are the parameters (heights) to estimate. $I_k(t)$ indicates if t belongs to the kth interval according to `c(0,knots,maxRange)`, where $k = 0$ indicates the interval `c(0,knots[1])`.

`tiaf.exponential`: $g(t|\kappa) = \exp(-\alpha_\kappa t)$,
which is the kernel of the exponential distribution. If `nTypes=1` (single-type epidemic or type-invariant `tiaf` in multi-type epidemic), then $\alpha_\kappa = \alpha$ for all types κ.

Value

The specification of an interaction function, which is a list. See `siaf` and `tiaf`, respectively, for a description of its components.

Author(s)

Sebastian Meyer

References

Meyer, S. and Held, L. (2014): Power-law models for infectious disease spread. *The Annals of Applied Statistics*, **8** (3), 1612-1639.
DOI-Link: http://dx.doi.org/10.1214/14-AOAS743

Meyer, S., Elias, J. and Höhle, M. (2012): A space-time conditional intensity model for invasive meningococcal disease occurrence. *Biometrics*, **68**, 607-616.
DOI-Link: http://dx.doi.org/10.1111/j.1541-0420.2011.01684.x

Meyer, S. (2010): Spatio-Temporal Infectious Disease Epidemiology based on Point Processes. Master's Thesis, Ludwig-Maximilians-Universität München.
Available as http://epub.ub.uni-muenchen.de/11703/

See Also

`twinstim`, `siaf`, `tiaf`

Examples

```
# constant temporal dispersal
tiaf.constant()
# step function kernel
tiaf.step(c(3,7), maxRange=14, nTypes=2)
# exponential decay specification
tiaf.exponential()

# Type-dependent Gaussian spatial interaction function using an adaptive
# two-dimensional midpoint-rule to integrate it over polygonal domains
siaf.gaussian(2, F.adaptive=TRUE)

# Type-independent power-law kernel
siaf.powerlaw()
```

```
# "lagged" power-law
siaf.powerlawL()

# (reparametrized) t-kernel
siaf.student()

# step function kernel
siaf.step(c(10,20,50), maxRange=100)
```

`twinstim_iafplot` *Plot the spatial or temporal interaction function of a twimstim*

Description

The function plots the fitted temporal or (isotropic) spatial interaction function of a `twinstim` object.

Usage

```
iafplot(object, which = c("siaf", "tiaf"), types = NULL,
        scaled = TRUE, truncated = FALSE, log="",
        conf.type = if (length(pars) > 1) "MC" else "parbounds",
        conf.level = 0.95, conf.B = 999, xgrid = 101,
        col.estimate = rainbow(length(types)), col.conf = col.estimate,
        alpha.B = 0.15, lwd = c(3,1), lty = c(1,2),
        verticals = FALSE, do.points = FALSE,
        add = FALSE, xlim = NULL, ylim = NULL, xlab = NULL, ylab = NULL,
        legend = !add && (length(types) > 1), ...)
```

Arguments

`object`	object of class `"twinstim"` containing the fitted model.
`which`	argument indicating which of the two interaction functions to plot. Possible values are `"siaf"` (default) for the spatial interaction function and `"tiaf"` for the temporal interaction function.
`types`	integer vector indicating for which event `types` the interaction function should be plotted in case of a marked twinstim. The default `types=NULL` checks if the interaction function is type-specific: if so, `types=1:nrow(object$qmatrix)` is used, otherwise `types=1`.
`scaled`	logical indicating if the interaction function should be multiplied by the epidemic intercept $exp(\gamma_0)$. This is the default and required for the comparison of estimated interaction functions from different models.
`truncated`	logical indicating if the plotted interaction function should take the maximum range of interaction (`eps.t`/`eps.s`) into account, i.e., drop to zero at that point (if it is finite after all). If there is no common range of interaction, a `rug` indicating the various ranges will be added to the plot if `truncated=TRUE`. If `truncated` is a scalar, this value is used as the point `eps` where the function drops to 0.

log — a character string passed to plot.default indicating which axes should be logarithmic. If add=TRUE, log is set according to par("xlog") and par("ylog").

conf.type — type of confidence interval to produce.
If conf.type="MC" (or "bootstrap"), conf.B parameter vectors are sampled from the asymptotic (multivariate) normal distribution of the ML estimate of the interaction function parameters; the interaction function is then evaluated on the xgrid (i.e. temporal or spatial distances from the host) for each parameter realization to obtain a conf.level confidence interval at each point of the xgrid (or to plot the interaction functions of all Monte-Carlo samples if conf.level=NA). Note that the resulting plot is .Random.seed-dependent for the Monte-Carlo type of confidence interval.
If conf.type="parbounds", the conf.level Wald confidence intervals for the interaction function parameters are calculated and the interaction function is evaluated on the xgrid (distances from the host) for all combinations of the bounds of the parameters and the point-wise extremes of those functions are plotted. This type of confidence interval is only valid in case of a single parameter, i.e. scaled + nsiafpars == 1, but could also be used as a rough indication if the Monte-Carlo approach takes too long. A warning is thrown if the "parbounds" type is used for multiple parameters.
If conf.type="none" or NA or NULL, no confidence interval will be calculated.

conf.level — the confidence level required. For conf.type = "MC" it may also be specified as NA, in which case all conf.B sampled functions will be plotted with transparency value given by alpha.B.

conf.B — number of samples for the "MC" (Monte Carlo) confidence interval.

xgrid — either a numeric vector of x-values (distances from the host) where to evaluate which, or a scalar representing the desired number of evaluation points in the interval c(0,xlim[2]).
If the interaction function is a step function (siaf.step or tiaf.step), xgrid is ignored and internally set to c(0, knots).

col.estimate — vector of colours to use for the function point estimates of the different types.

col.conf — vector of colours to use for the confidence intervals of the different types.

alpha.B — alpha transparency value (as relative opacity) used for the conf.B sampled interaction functions in case conf.level = NA

lwd, lty — numeric vectors of length two specifying the line width and type of point estimates (first element) and confidence limits (second element), respectively.

verticals,do.points — graphical settings for step function kernels. These can be logical (as in plot.stepfun) or lists of graphical parameters.

add — add to an existing plot?

xlim, ylim — vectors of length two containing the x- and y-axis limit of the plot. The default y-axis range (ylim=NULL) is from 0 to 1 (or to $exp(\gamma_0)$, if scaled). The default x-axis (xlim=NULL) starts at 0, and the upper limit is determined as follows (in decreasing order of precedence):

- If xgrid is a vector of evaluation points, xlim[2] is set to max(xgrid).
- eps.t/eps.s if it is unique and finite.

- If the interaction function is a step function with `maxRange<Inf`, i.e. it drops to 0 at `maxRange`, `xlim[2]` is set to `maxRange`.
- Otherwise, it is set to the length of the observation period (`which="tiaf"`) or the diagonale length of the bounding box of the observation region (`which="siaf"`), respectively.

`xlab, ylab`	labels for the axes with `NULL` providing sensible defaults.
`legend`	logical indicating if a legend for the `types` should be added. It can also be a list of arguments passed to `legend` to tweak the default settings.
`...`	additional arguments passed to the default `plot` method.

Value

A plot is created – see e.g. Figure 3(b) in Meyer et al. (2012).

The function invisibly returns a matrix of the plotted values of the interaction function (evaluated on `xgrid`, by type). The first column of the matrix contains the distance x, and the remaining `length(types)` columns contain the (scaled) function values for each type.

The pointwise confidence intervals of the interaction functions are returned in similar matrices as attributes: if `length(types)==1`, there is a single attribute `"CI"`, whereas for multiple types, the attributes are named `paste0("CI.",typeNames)` (where the `typeNames` are retrieved from `object$qmatrix`).

Author(s)

Sebastian Meyer

References

Meyer, S., Elias, J. and Höhle, M. (2012): A space-time conditional intensity model for invasive meningococcal disease occurrence. *Biometrics*, **68**, 607-616.
DOI-Link: http://dx.doi.org/10.1111/j.1541-0420.2011.01684.x

See Also

`plot.twinstim`, which calls this function.

Examples

```
data("imdepifit")

iafplot(imdepifit, "tiaf", scaled=FALSE)   # tiaf.constant(), not very exciting
iafplot(imdepifit, "siaf", scaled=FALSE)

# scaled version uses a Monte-Carlo-CI
set.seed(1)  # result depends on .Random.seed
iafplot(imdepifit, "siaf", scaled=TRUE, conf.type="MC", conf.B=199,
        col.conf=gray(0.4), conf.level=NA)  # show MC samples
```

twinstim_intensity *Plotting Intensities of Infection over Time or Space*

Description

intensityplot method to plot the evolution of the total infection intensity, its epidemic proportion or its endemic proportion over time or space (integrated over the other dimension) of fitted twinstim models (or simEpidataCS). The "simEpidataCS"-method is just a wrapper around intensityplot.twinstim by making the "simEpidataCS" object "twinstim"-compatible, i.e. enriching it by the required model components and environment.

The intensity.twinstim auxiliary function returns functions which calculate the endemic or epidemic intensity at a specific time point or location (integrated over the other dimension).

Usage

```
## S3 method for class 'twinstim'
intensityplot(x,
    which = c("epidemic proportion", "endemic proportion", "total intensity"),
    aggregate = c("time", "space"), types = 1:nrow(x$qmatrix),
    tiles, tiles.idcol = NULL, plot = TRUE, add = FALSE,
    tgrid = 101, rug.opts = list(),
    sgrid = 128, polygons.args = list(), points.args = list(),
    cex.fun = sqrt, ...)

## S3 method for class 'simEpidataCS'
intensityplot(x, ...)

intensity.twinstim(x,
    aggregate = c("time", "space"), types = 1:nrow(x$qmatrix),
    tiles, tiles.idcol = NULL)
```

Arguments

x	an object of class "twinstim" or "simEpidataCS", respectively.
which	"epidemic proportion", "endemic proportion", or "total intensity". Partial matching is applied. Determines whether to plot the path of the total intensity or its epidemic or endemic proportions over time or space (which) aggregated over the other dimension and types.
aggregate	One of "time" or "space". The former results in a plot of the evolution of which as a function of time (integrated over the observation region $\boldsymbol{W}$), whereas the latter produces a spplot of which over $\boldsymbol{W}$ (spanned by tiles). In both cases, which is evaluated on a grid of values, given by tgrid or sgrid, respectively.
types	event types to aggregate. By default, all types of events are aggregated, but one could also be interested in only one specific type or a subset of event types.

`tiles` object of class `SpatialPolygons` representing the decomposition of $\boldsymbol{W}$ into different regions (as used in the corresponding `stgrid` of the `"epidataCS"`. This is only needed for `aggregate = "space"`.

`tiles.idcol` either a column index for `tiles@data` (if `tiles` is a `SpatialPolygonsDataFrame`), or `NULL` (default), which refers to the `"ID"` slot of the polygons, i.e., `row.names(tiles)`. The ID's must correspond to the factor levels of `stgrid$tile` of the `"epidataCS"` on which `x` was fitted.

`plot` logical indicating if a plot is desired, which defaults to `TRUE`. Otherwise, a function will be returned, which takes a vector of time points (if `aggregate = "time"`) or a matrix of coordinates (if `aggregate = "space"`), and returns `which` on this grid.

`add` logical. If `TRUE` and `aggregate = "time"`, paths are added to the current plot, using `lines`. This does not work for `aggregate = "space"`.

`tgrid` either a numeric vector of time points when to evaluate `which`, or a scalar representing the desired number of evaluation points in the observation interval $[t_0, T]$. This argument is unused for `aggregate = "space"`.

`rug.opts` if a list, its elements are passed as arguments to the function `rug`, which will mark the time points of the events if `aggregate = "time"` (it is unused in the spatial case); otherwise (e.g., `NULL`), no `rug` will be produced. By default, the `rug` argument `ticksize` is set to 0.02 and `quiet` is set to `TRUE`. Note that the argument `x` of the `rug` function, which contains the locations for the `rug` is fixed internally and can not be modified.

`sgrid` either an object of class `"SpatialPixels"` (or coercible to that class) representing the locations where to evaluate `which`, or a scalar representing the total number of points of a grid constructed on the bounding box of `tiles` (using `Sobj_SpatialGrid` from package **maptools**). `sgrid` is internally subsetted to contain only points inside `tiles`. This argument is unused for `aggregate = "time"`.

`polygons.args` if a list, its elements are passed as arguments to `sp.polygons`, which will add `tiles` to the plot if `aggregate = "space"` (it is unused for the temporal plot). By default, the fill `colour` of the tiles is set to `"darkgrey"`.

`points.args` if a list, its elements are passed as arguments to `sp.points`, which will add the event locations to the plot if `aggregate = "space"` (it is unused for the temporal plot). By default, the plot symbol ist set to `pch=1`. The sizes of the points are determined as the product of the argument `cex` (default: 0.5) of this list and the sizes obtained from the function `cex.fun` which accounts for multiple events at the same location.

`cex.fun` function which takes a vector of counts of events at each unique location and returns a (vector of) `cex` value(s) for the sizes of the points at the event locations used in `points.args`. Defaults to the `sqrt()` function, which for the default circular `pch=1` means that the area of each point is proportional to the number of events at its location.

`...` further arguments passed to `plot` or `lines` (if `aggregate = "time"`), or to `spplot` (if `aggregate = "space"`).
For `intensityplot.simEpidataCS`, arguments passed to `intensityplot.twinstim`.

Value

If `plot = FALSE` or `aggregate = "time"`, a function is returned, which takes a vector of time points (if `aggregate = "time"`) or a matrix of coordinates (if `aggregate = "space"`), and returns `which` on this grid.

If `plot = TRUE` and `aggregate = "space"`, the `trellis.object` containg the spatial plot is returned.

Author(s)

Sebastian Meyer

See Also

`plot.twinstim`, which calls this function.

Examples

```
data("imdepi")
data("imdepifit")

# for the intensityplot we need the model environment, which can be
# easily added by the intelligent update method (no need to refit the model)
imdepifit <- update(imdepifit, model=TRUE)

## path of the total intensity
opar <- par(mfrow=c(2,1))
intensityplot(imdepifit, which="total intensity",
              aggregate="time", tgrid=500)
plot(imdepi, "time", breaks=100)
par(opar)

## time course of the epidemic proportion by event
intensityplot(imdepifit, which="epidemic proportion",
              aggregate="time", tgrid=500, types=1)
intensityplot(imdepifit, which="epidemic proportion",
              aggregate="time", tgrid=500, types=2, add=TRUE, col=2)
legend("topright", legend=levels(imdepi$events$type), lty=1, col=1:2,
       title = "event type")

## spatial shape of the intensity (aggregated over time)
if (surveillance.options("allExamples") && requireNamespace("maptools"))
{
  ## load borders of Germany's districts
  load(system.file("shapes", "districtsD.RData", package="surveillance"))

  # total intensity (using a rather sparse 'sgrid' for speed)
  intensityplot(imdepifit, which="total intensity",
                aggregate="space", tiles=districtsD, sgrid=500)

  # epidemic proportion by type
  maps_epiprop <- lapply(1:2, function (type) {
```

```
      intensityplot(imdepifit, which="epidemic", aggregate="space",
                    types=type, tiles=districtsD, sgrid=1000,
                    at=seq(0,1,by=0.1), col.regions=rev(heat.colors(20)))
  })
  plot(maps_epiprop[[1]], split=c(1,1,2,1), more=TRUE)
  plot(maps_epiprop[[2]], split=c(2,1,2,1))
}
```

---

twinstim_methods *Print, Summary and Extraction Methods for "twinstim" Objects*

Description

Besides print and summary methods there are also some standard extraction methods defined for objects of class "twinstim": vcov, logLik, and nobs. This also enables the use of, e.g., confint and AIC. The model summary can be exported to LaTeX by the corresponding toLatex or xtable methods.

Usage

```
## S3 method for class 'twinstim'
print(x, digits = max(3, getOption("digits") - 3), ...)
## S3 method for class 'twinstim'
summary(object, test.iaf = FALSE,
        correlation = FALSE, symbolic.cor = FALSE, ...)

## S3 method for class 'twinstim'
vcov(object, ...)
## S3 method for class 'twinstim'
logLik(object, ...)
## S3 method for class 'twinstim'
nobs(object, ...)

## S3 method for class 'summary.twinstim'
print(x,
      digits = max(3, getOption("digits") - 3), symbolic.cor = x$symbolic.cor,
      signif.stars = getOption("show.signif.stars"), ...)

## S3 method for class 'summary.twinstim'
toLatex(object,
        digits = max(3, getOption("digits") - 3), eps.Pvalue = 1e-4,
        align = "lrrrr", booktabs = getOption("xtable.booktabs", FALSE),
        withAIC = FALSE, ...)
## S3 method for class 'summary.twinstim'
xtable(x, caption = NULL, label = NULL,
      align = c("l", "r", "r", "r"), digits = 3,
      display = c("s", "f", "s", "s"),
      ci.level = 0.95, ci.fmt = "%4.2f", ci.to = "--",
      eps.Pvalue = 1e-4, ...)
```

Arguments

x, object	an object of class "twinstim" or "summary.twinstim", respectively.
digits	integer, used for number formatting with signif(). Minimum number of significant digits to be printed in values.
test.iaf	logical indicating if the simple Wald z- and p-values should be calculated for parameters of the interaction functions siaf and tiaf. Because it is often invalid or meaningless to do so, the default is FALSE.
correlation	logical. If TRUE, the correlation matrix of the estimated parameters is returned and printed.
symbolic.cor	logical. If TRUE, print the correlations in a symbolic form (see symnum) rather than as numbers.
signif.stars	logical. If TRUE, "significance stars" are printed for each coefficient.
eps.Pvalue	passed to format.pval.
booktabs	logical indicating if the toprule, midrule and bottomrule commands from the LaTeX package **booktabs** should be used for horizontal lines rather than hline.
withAIC	logical indicating if the AIC and the log-likelihood of the model should be included below the table of coefficients in the LaTeX tabular.
caption,label,align,display	see xtable.
ci.level,ci.fmt,ci.to	the confidence intervals are calculated at level ci.level and printed using sprintf with format ci.fmt and seperator ci.to.
...	For print.summary.twinstim, arguments passed to printCoefmat. For all other methods: unused (argument of the generic).

Details

The estimated coefficients and standard Wald-type confidence intervals can be extracted using the default coef and confint methods from package **stats**.

The print and summary methods allow the compact or comprehensive representation of the fitting results, respectively. The former only prints the original function call, the estimated coefficients and the maximum log-likelihood value. The latter prints the whole coefficient matrix with standard errors, z- and p-values (see printCoefmat) – separately for the endemic and the epidemic component – and additionally the AIC, the achieved log-likelihood, the number of log-likelihood and score evaluations, and the runtime. They both append a big "WARNING", if the optimization algorithm did not converge.

The toLatex method is essentially a translation of the printed summary table of coefficients to LaTeX code (using **xtable**). However, the xtable method does a different job in that it first converts coefficients to rate ratios (RR, i.e., the exp-transformation) and gives confidence intervals for those instead of standard errors and z-values. Intercepts and interaction function parameters are ignored by the xtable method.

Value

The `print` methods return their first argument, invisibly, as they always should. The `vcov` method returns the estimated variance-covariance matrix of the parameters, which is the inverse of `object$fisherinfo` (estimate of the *expected* Fisher information matrix). This `"fisherinfo"` is not always available (see `twinstim`), in which case `object$fisherinfo.observed` is used if available or an error is returned otherwise. The `logLik` and `nobs` methods return the maximum log-likelihood value of the model, and the number of events (excluding events of the pre-history), respectively.

The `summary` method returns a list containing some summary statistics of the model, which is nicely printed by the corresponding `print` method.

The `toLatex` method returns a character vector of class `"Latex"`, each element containing one line of LaTeX code (see `print.Latex`). The `xtable` method returns an object of class `"xtable"`. Note that the column name of the confidence interval, e.g. "95% CI", contains the percent symbol that may need to be escaped when printing the `"xtable"` in the output format (see `sanitize.text.function` in `print.xtable`). This may also hold for row names.

Author(s)

Sebastian Meyer

Examples

```
# load a fit of the 'imdepi' data, see the example in ?twinstim
data("imdepifit")

# print method
imdepifit

# extract point estimates
coef(imdepifit)

# variance-covariance matrix of endemic parameters
# (inverse of expected Fisher information)
unname(vcov(imdepifit)[1:4,1:4])

# the default confint() method may be used for Wald CI's
confint(imdepifit, parm="e.typeC", level=0.95)

# log-likelihood and AIC of the fitted model
logLik(imdepifit)
AIC(imdepifit)
nobs(imdepifit)

# summary method
summary(imdepifit, test.iaf=FALSE, correlation=TRUE, symbolic.cor=TRUE)

# create LaTeX code of coefficient table
toLatex(summary(imdepifit), withAIC=FALSE)

# or using the xtable-method (which produces rate ratios)
xtable(summary(imdepifit))
```

twinstim_plot *Plot methods for fitted* twinstim*'s*

Description

The fitted conditional intensity function from twinstim may be visualized in at least two ways: iafplot plots the fitted interaction functions (as a function of the distance from the host), and intensityplot.twinstim plots the fitted intensity either aggregated over space (evolution over time) or aggregated over time (spatial surface of the cumulated intensity). The plot method for class "twinstim" is just a wrapper for these two functions.

Usage

```
## S3 method for class 'twinstim'
plot(x, which, ...)
```

Arguments

x	an object of class "twinstim".
which	character. Which characteristic of the conditional intensity should be plotted? Possible values are the ones allowed in the functions iafplot and intensityplot.twinstim, e.g. "siaf", or "epidemic proportion". Partial matching is applied.
...	further arguments passed to iafplot or intensityplot.twinstim.

Value

See the documentation of the respective plot functions, iafplot or intensityplot.twinstim.

Author(s)

Sebastian Meyer

Examples

```
# see the examples for iafplot() and intensityplot.twinstim()
```

twinstim_profile *Profile Likelihood Computation and Confidence Intervals for* twinstim *objects*

Description

Function to compute estimated and profile likelihood based confidence intervals for twinstim objects. Computations might be cumbersome!

WARNING: the implementation is not well tested, simply uses optim (ignoring optimizer settings from the original fit), and does not return the complete set of coefficients at each grid point.

Usage

```
## S3 method for class 'twinstim'
profile(fitted, profile, alpha = 0.05,
        control = list(fnscale = -1, factr = 10, maxit = 100),
        do.ltildeprofile=FALSE, ...)
```

Arguments

`fitted`	an object of class `"twinstim"`.
`profile`	a list with elements being numeric vectors of length 4. These vectors must have the form `c(index, lower, upper, gridsize)`. `index`: index of the parameter to be profiled in the vector `coef(fitted)`. `lower, upper`: lower/upper limit of the grid on which the profile log-likelihood is evaluated. Can also be `NA` in which case `lower/upper` equals the lower/upper bound of the respective 0.3 % Wald confidence interval (+-3*se). `gridsize`: grid size of the equally spaced grid between lower and upper. Can also be 0 in which case the profile log-likelihood for this parameter is not evaluated on a grid.
`alpha`	$(1-\alpha)\%$ profile likelihood based confidence intervals are computed. If alpha <= 0, then no confidence intervals are computed. This is currently not implemented.
`control`	control object to use in `optim` for the profile log-likelihood computations. It might be necessary to control `maxit` or `reltol` in order to obtain results in finite time.
`do.ltildeprofile`	If `TRUE` calculate profile likelihood as well. This might take a while, since an optimisation for all other parameters has to be performed. Useful for likelihood based confidence intervals. Default: `FALSE`.
`...`	unused (argument of the generic).

Value

list with profile log-likelihood evaluations on the grid and highest likelihood and wald confidence intervals. The argument `profile` is also returned.

Author(s)

Michael Höhle

Examples

```
# the following call takes a while
## Not run:
#Load the twinstim model fitted to the IMD data
data("imdepifit")
# for profiling we need the model environment
imdepifit <- update(imdepifit, model=TRUE)

#Generate profiling object for a list of parameters for the new model
```

```
names <- c("h.(Intercept)","e.typeC")
coefList <- lapply(names, function(name) {
  c(pmatch(name,names(coef(imdepifit))),NA,NA,11)
})

#Profile object (necessary to specify a more loose convergence
#criterion). Speed things up by using do.ltildeprofile=FALSE (the default)
prof <- profile(imdepifit, coefList,control=list(fnscale=-1,maxit=50,
   reltol=0.1,REPORT=1,trace=5),do.ltildeprofile=TRUE)

#Plot result for one variable
par(mfrow=c(1,2))
for (name in names) {
  with(as.data.frame(prof$lp[[name]]),matplot(grid,cbind(profile,estimated,wald),
    type="l",xlab=name,ylab="loglik"))
  legend(x="bottomleft",c("profile","estimated","wald"),lty=1:3,col=1:3)
}

## End(Not run)
```

`twinstim_siaf` *Spatial Interaction Function Objects*

Description

A spatial interaction function for use in `twinstim` can be constructed via the `siaf` function. It checks the supplied function elements, assigns defaults for missing arguments, and returns all checked arguments in a list. However, for standard applications it is much easier to use one of the pre-defined spatial interaction functions, e.g., `siaf.gaussian`.

Usage

```
siaf(f, F, Fcircle, effRange, deriv, Deriv, simulate, npars,
     validpars = NULL)
```

Arguments

`f`	the spatial interaction function. It must accept two arguments, the first one being a (2-column) coordinate matrix, the second one a parameter vector. For marked `twinstim`, it must accept the type of the event (integer code) as its third argument (either a single type for all locations or separate types for each location).
`F`	function computing the integral of $f(s)$ (passed as second argument) over a polygonal `"owin"` domain (first argument). The third and fourth argument are the parameter vector and the (*single*) type, respectively. There may be additional arguments, which can then be specified in the `control.siaf$F` argument list of `twinstim`. If the `F` function is missing, a general default (`polyCub`) will be used, with extra arguments `method` (default: `"SV"`) and corresponding accuracy parameters.

`Fcircle` optional function for fast calculation of the (two-dimensional) integral of $f(s)$ over a circle with radius `r` (first argument). Further arguments are as for `f`. It must not be vectorized (will always be called with single radius and a single type). If this function is specified, integration of the `siaf` over the spatial influence region of an event will be faster if the region is actually circular. This is the case if the event is located at least a distance `eps.s` from the border of the observation region `W`, or if the distance to the border is larger than the effective integration range (if specified, see `effRange` below).

`effRange` optional function returning the "effective" range of $f(s)$ for the given set of parameters (the first and only argument) such that the circle with radius `effRange` contains the numerically essential proportion of the integral mass. For the Gaussian kernel the default is `function (logsd) 6*exp(logsd)`. The return value must be a vector of length `nTypes` (effective range for each type). This function is only used if `Fcircle` is also specified.

`deriv` optional derivative of $f(s)$ *with respect to the parameters*. It takes the same arguments as `f` but returns a matrix with as many rows as there were coordinates in the input and `npars` columns. This derivative is necessary for the calculation of the score function in `twinstim()`, which is advantageous for the numerical log-likelihood maximization.

`Deriv` function computing the integral of `deriv` (passed as second argument) over a polygonal `"owin"` domain (first argument). The return value is thus a vector of length `npars`. The third argument is the parameter vector and the fourth argument is a (*single*) type and must be named `type`. There may be additional arguments, which can then be specified in the `control.siaf$Deriv` argument list of `twinstim`. If the `Deriv` function is missing, a general default (`polyCub`) will be used, with extra arguments `method` (default: `"SV"`) and corresponding accuracy parameters.

`simulate` optional function returning a sample drawn from the spatial kernel (only required for the simulation of `twinstim` models). Its first argument is the size of the sample to generate, next the parameter vector, an optional single event type, and an optional upperbound for the radius within which to simulate points. The function must return a two-column *matrix* of the sampled locations. Note that the simulation method actually samples only one location at a time, thus it is sufficient to have a working `function(n=1, pars, type, ub)`.

`npars` the number of parameters of the spatial interaction function `f` (i.e. the length of its second argument).

`validpars` optional function taking one argument, the parameter vector, indicating if it is valid. This approach to specify parameter constraints is rarely needed, because usual box-constrained parameters can be taken into account by using L-BFGS-B as the optimization method in `twinstim` (with arguments `lower` and `upper`), and positivity constraints by using log-parametrizations. This component is not necessary (and ignored) if `npars == 0`.

Value

list of checked arguments.

Author(s)

Sebastian Meyer

See Also

siaf.gaussian for a pre-defined spatial interaction function, and tiaf for the temporal interaction function.

---

twinstim_simulation *Simulation of a Self-Exciting Spatio-Temporal Point Process*

Description

The function simEpidataCS simulates events of a self-exiciting spatio-temporal point process of the "twinstim" class. Simulation works via Ogata's modified thinning of the conditional intensity as described in Meyer et al. (2012). Note that simulation is limited to the spatial and temporal range of stgrid.

The simulate method for objects of class "twinstim" simulates new epidemic data using the model and the parameter estimates of the fitted object.

Usage

```
simEpidataCS(endemic, epidemic, siaf, tiaf, qmatrix, rmarks,
    events, stgrid, tiles, beta0, beta, gamma, siafpars, tiafpars,
    t0 = stgrid$start[1], T = tail(stgrid$stop,1), nEvents = 1e5,
    control.siaf = list(F=list(), Deriv=list()),
    W = NULL, trace = 5, nCircle2Poly = 32, gmax = NULL, .allocate = 500,
    .skipChecks = FALSE, .onlyEvents = FALSE)

## S3 method for class 'twinstim'
simulate(object, nsim = 1, seed = NULL, data, tiles,
    rmarks = NULL, t0 = NULL, T = NULL, nEvents = 1e5,
    control.siaf = object$control.siaf,
    W = data$W, trace = FALSE, nCircle2Poly = NULL, gmax = NULL,
    .allocate = 500, simplify = TRUE, ...)
```

Arguments

endemic	see twinstim. Note that type-specific endemic intercepts are specified by beta0 here, not by the term (1|type).
epidemic	see twinstim. Marks appearing in this formula must be returned by the generating function rmarks.
siaf	see twinstim. In addition to what is required for fitting with twinstim, the siaf specification must also contain the element simulate, a function which draws random locations following the spatial kernel siaf$f. The first argument of the function is the number of points to sample (say n), the second one is the

vector of parameters siafpars, the third one is the type indicator (a character string matching a type name as specified by dimnames(qmatrix)). With the current implementation there will always be simulated only one location at a time, i.e. n=1. The predefined siaf's all provide simulation.

tiaf
: e.g. what is returned by the generating function tiaf.constant or tiaf.exponential. See also twinstim.

qmatrix
: see epidataCS. Note that this square matrix and its dimnames determine the number and names of the different event types. In the simplest case, there is only a single type of event, i.e. qmatrix = diag(1).

rmarks
: function of single time (1st arg) and location (2nd arg) returning a one-row data.frame of marks (named according to the variables in epidemic) for an event at this point. This must include the columns eps.s and eps.t, i.e. the values of the spatial and temporal interaction ranges at this point. Only "numeric" and "factor" columns are allowed. Assure that factor variables are coded equally (same levels and level order) for each new sample.

  For the simulate.twinstim method, the default (NULL) means sampling from the empirical distribution function of the (non-missing) marks in data restricted to events in the simulation period (t0;T]. If there are no events in this period, e.g., if simulating beyond the original observation period, rmarks will sample marks from all of data$events.

events
: NULL or missing (default) in case of an empty prehistory, or a SpatialPointsDataFrame containing events of the prehistory (-Inf;t0] of the process (required for the epidemic to start in case of no endemic component in the model). The SpatialPointsDataFrame must have the same proj4string as tiles and W). The attached data.frame (data slot) must contain the typical columns as described in as.epidataCS (time, tile, eps.t, eps.s, and, for type-specific models, type) and all marks appearing in the epidemic specification. Note that some column names are reserved (see as.epidataCS). Only events up to time t0 are selected and taken as the prehistory.

stgrid
: see as.epidataCS. Simulation only works inside the spatial and temporal range of stgrid.

tiles
: object inheriting from "SpatialPolygons" with row.names matching the tile names in stgrid and having the same proj4string as events and W. This is necessary to sample the spatial location of events generated by the endemic component.

beta0,beta,gamma,siafpars,tiafpars
: these are the parameter subvectors of the twinstim. beta and gamma must be given in the same order as they appear in endemic and epidemic, respectively. beta0 is either a single endemic intercept or a vector of type-specific endemic intercepts in the same order as in qmatrix.

t0
: events having occured during (-Inf;t0] are regarded as part of the prehistory H_0 of the process. The time point t0 must be an element of stgrid$start. For simEpidataCS, by default, and also if t0=NULL, it is the earliest time point of the spatio-temporal grid stgrid. For the simulate.twinstim method, NULL means to use the same time range as for the fitting of the "twinstim" object.

`T, nEvents` simulate a maximum of `nEvents` events up to time `T`, then stop. For `simEpidataCS`, by default, and also if `T=NULL`, `T` equals the last stop time in `stgrid` (it cannot be greater) and `nEvents` is bounded above by 10000. For the `simulate.twinstim` method, `T=NULL` means to use the same same time range as for the fitting of the `"twinstim" object`.

`W` see `as.epidataCS`. When simulating from `twinstim`-fits, `W` is by default taken from the original `data$W`. If specified as `NULL`, `W` is generated automatically via `unionSpatialPolygons(tiles)`. However, since the result of such a polygon operation should always be verified, it is recommended to do that in advance. It is important that `W` and `tiles` cover the same region: on the one hand direct offspring is sampled in the spatial influence region of the parent event, i.e., in the intersection of `W` and a circle of radius the `eps.s` of the parent event, after which the corresponding tile is determined by overlay with `tiles`. On the other hand endemic events are sampled from `tiles`.

`trace` logical (or integer) indicating if (or how often) the current simulation status should be cated. For the `simulate.twinstim` method, `trace` currently only applies to the first of the `nsim` simulations.

`.allocate` number of rows (events) to initially allocate for the event history; defaults to 500. Each time the simulated epidemic exceeds the allocated space, the event `data.frame` will be enlarged by `.allocate` rows.

`.skipChecks,.onlyEvents` these logical arguments are not meant to be set by the user. They are used by the simulate-method for twinstim objects.

`object` an object of class `"twinstim"`.

`nsim` number of epidemics (i.e. spatio-temporal point patterns inheriting from class `"epidataCS"`) to simulate. Defaults to 1 when the result is a simple object inheriting from class `"simEpidataCS"` (as if `simEpidataCS` would have been called directly). If `nsim > 1`, the result will be a list the structure of which depends on the argument `simplify`.

`seed` an object specifying how the random number generator should be initialized for simulation (via `set.seed`). The initial state will also be stored as an attribute `"seed"` of the result. The original state of the `.Random.seed` will be restored at the end of the simulation. By default (`NULL`), neither initialization nor recovery will be done. This behaviour is copied from the `simulate.lm` method.

`data` an object of class `"epidataCS"`, usually the one to which the `"twinstim" object` was fitted. It carries the `stgrid` of the endemic component, but also `events` for use as the prehistory, and defaults for `rmarks` and `nCircle2Poly`.

`simplify` logical. It is strongly recommended to set `simplify = TRUE` (default) if `nsim` is large. This saves space and computation time, because for each simulated epidemic only the `events` component is saved. All other components, which do not vary between simulations, are only stored from the first run. In this case, the runtime of each simulation is stored as an attribute `"runtime"` to each simulated `events`. See also the "Value" section below.

`control.siaf` see `twinstim`.

`nCircle2Poly` see `as.epidataCS`. For `simulate.twinstim`, `NULL` means to use the same value as for `data`.

gmax — maximum value the temporal interaction function `tiaf$g` can attain. If `NULL`, then it is assumed as the maximum value of the type-specific values at 0, i.e. `max(tiaf$g(rep.int(0,nTypes), tiafpars, 1:nTypes))`.

... — unused (arguments of the generic).

Value

The function `simEpidataCS` returns a simulated epidemic of class `"simEpidataCS"`, which enhances the class `"epidataCS"` by the following additional components known from objects of class `"twinstim"`: `timeRange`, `formula`, `coefficients`, `npars`, `call`, `runtime`.

The `simulate.twinstim` method has some additional *attributes* set on its result: `call`, `seed`, `simplified`, and `runtime` with their obvious meanings. Furthermore, if `nsim > 1`, it returns an object of class `"simEpidataCSlist"`, the form of which depends on the value of `simplify`: if `simplify = FALSE`, then the return value is just a list of sequential simulations, each of class `"simEpidataCS"`. However, if `simplify = TRUE`, then the sequential simulations share all components but the simulated `events`, i.e. the result is a list with the same components as a single object of class `"simEpidataCS"`, but with `events` replaced by an `eventsList` containing the `events` returned by each of the simulations.

The `stgrid` component of the returned `"simEpidataCS"` will be truncated to the actual end of the simulation, which might be $< T$, if the upper bound `nEvents` is reached during simulation.

CAVE: Currently, `simplify=TRUE` in `simulate.twinstim` ignores that multiple simulated epidemics (`nsim > 1`) may have different `stgrid` time ranges. In a `"simEpidataCSlist"`, the `stgrid` shared by all of the simulated epidemics is just the `stgrid` returned by the *first* simulation.

Note

The more detailed the polygons in `tiles` are the slower is the algorithm. Often it can be advantageous to sacrifice some shape detail for speed by reducing polygon complexity using, e.g., the Douglas and Peucker (1973) reduction method available at `MapShaper.org` (Harrower and Bloch, 2006) or as function `thinnedSpatialPoly` in package **maptools**, or by passing via **spatstat**'s `simplify.owin` procedure.

Author(s)

Sebastian Meyer, with contributions by Michael Höhle

References

Douglas, D. H. and Peucker, T. K. (1973): Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, **10**, 112-122

Harrower, M. and Bloch, M. (2006): MapShaper.org: A Map Generalization Web Service. *IEEE Computer Graphics and Applications*, **26**(4), 22-27.
DOI-Link: http://dx.doi.org/10.1109/MCG.2006.85

Meyer, S., Elias, J. and Höhle, M. (2012): A space-time conditional intensity model for invasive meningococcal disease occurrence. *Biometrics*, **68**, 607-616.
DOI-Link: http://dx.doi.org/10.1111/j.1541-0420.2011.01684.x

Meyer, S. (2010): Spatio-Temporal Infectious Disease Epidemiology based on Point Processes. Master's Thesis, Ludwig-Maximilians-Universität München.
Available as http://epub.ub.uni-muenchen.de/11703/

See Also

The plot.epidataCS and animate.epidataCS methods for plotting and animating continuous-space epidemic data, respectively, which also work for simulated epidemics (by inheritance).

Function twinstim for fitting spatio-temporal conditional intensity models to epidemic data.

Examples

```
data("imdepi")
data("imdepifit")

## load borders of Germany's districts (originally obtained from the
## Bundesamt für Kartographie und Geodäsie, Frankfurt am Main, Germany,
## www.geodatenzentrum.de), simplified by the "modified Visvalingam"
## algorithm (level=6.6%) using MapShaper.org (v. 0.1.17):
load(system.file("shapes", "districtsD.RData", package="surveillance"))
plot(districtsD)
plot(stateD, add=TRUE, border=2, lwd=2)
# 'stateD' was obtained as 'rgeos::gUnaryUnion(districtsD)'

## simulate 2 realizations (during a VERY short period -- for speed)
## considering events from data(imdepi) before t=31 as pre-history
mysims <- simulate(imdepifit, nsim=2, seed=1, data=imdepi,
                   tiles=districtsD, t0=31, T=61, trace=FALSE,
                   nCircle2Poly=16, simplify=TRUE)

## extract the first realization -> object of class simEpidataCS
mysim1 <- mysims[[2]]
summary(mysim1)
plot(mysim1, aggregate="space")

## plot both epidemics using the plot-method for simEpidataCSlist's
plot(mysims, aggregate="time", subset=type=="B")


if (surveillance.options("allExamples")) {
### compare the observed _cumulative_ number of cases during the
### first 90 days to 20 simulations from the fitted model
### (performing these simulations takes about 30 seconds)

sims <- simulate(imdepifit, nsim=20, seed=1, data=imdepi, t0=0, T=90,
                 tiles=districtsD, simplify=TRUE)

## extract cusums
getcsums <- function (events) {
    tapply(events$time, events@data["type"],
           function (t) cumsum(table(t)), simplify=FALSE)
}
```

```
csums_observed <- getcsums(imdepi$events)
csums_simulated <- lapply(sims$eventsList, getcsums)

## plot it
plotcsums <- function (csums, ...) {
    mapply(function (csum, ...) lines(as.numeric(names(csum)), csum, ...),
           csums, ...)
    invisible()
}
plot(c(0,90), c(0,35), type="n", xlab="Time [days]",
     ylab="Cumulative number of cases")
plotcsums(csums_observed, col=c(2,4), lwd=3)
legend("topleft", legend=levels(imdepi$events$type), col=c(2,4), lwd=1)
invisible(lapply(csums_simulated, plotcsums,
                 col=scales::alpha(c(2,4), alpha=0.5)))
}


if (surveillance.options("allExamples")) {
### 'nsim' simulations of 'nm2add' months beyond the observed period:
nm2add <- 24
nsim <- 5
### With these settings, simulations will take about 30 seconds.
### The events still infective by the end of imdepi$stgrid will be used
### as the prehistory for the continued process.

origT <- tail(imdepi$stgrid$stop, 1)
## create a time-extended version of imdepi
imdepiext <- local({
    ## first we have to expand stgrid (assuming constant "popdensity")
    g <- imdepi$stgrid
    g$stop <- g$BLOCK <- NULL
    gadd <- data.frame(start=rep(seq(origT, by=30, length.out=nm2add),
                                 each=nlevels(g$tile)),
                       g[rep(seq_len(nlevels(g$tile)), nm2add), -1])
    ## now create an "epidataCS" using this time-extended stgrid
    as.epidataCS(events=imdepi$events,  # the replacement warnings are ok
                 W=imdepi$W, qmatrix=imdepi$qmatrix,
                 stgrid=rbind(g, gadd), T=max(gadd$start) + 30)
})
newT <- tail(imdepiext$stgrid$stop, 1)

## simulate beyond the original period
simsext <- simulate(imdepifit, nsim=nsim, seed=1, t0=origT, T=newT,
                    data=imdepiext, tiles=districtsD, simplify=TRUE)

## Aside to understand the note from checking events and tiles:
# marks(imdepi)["636",]  # tile 09662 is attributed to this event, but:
# plot(districtsD[c("09678","09662"),], border=1:2, lwd=2, axes=TRUE)
# points(imdepi$events["636",])
## this mismatch is due to polygon simplification

## plot the observed and simulated event numbers over time
```

```
## and ignore the warnings of plot.histogram on wrong area-representation
plot(imdepiext, breaks=c(unique(imdepi$stgrid$start),origT),
     cumulative=list(maxat=330), col=c(2,4))
for (i in seq_along(simsext$eventsList))
    plot(simsext[[i]], add=TRUE, legend.types=FALSE,
         breaks=c(unique(simsext$stgrid$start),newT),
         subset=!is.na(source),  # have to exclude the events of the prehistory
         cumulative=list(offset=c(table(imdepi$events$type)), maxat=330, axis=FALSE),
         col=scales::alpha(c(2,4), alpha=0.5))
abline(v=origT, lty=2, lwd=2)
}
```

`twinstim_step` *Stepwise Model Selection by AIC*

Description

`stepComponent` is a wrapper around `step` to select a `"twinstim"` component's model based on an information criterion in a stepwise algorithm.

There are also stand-alone single-step methods of `add1` and `drop1`.

Usage

```
stepComponent(object, component = c("endemic", "epidemic"),
              scope = list(upper = object$formula[[component]]),
              direction = "both", trace = 2, verbose = FALSE, ...)

## S3 method for class 'twinstim'
add1(object, scope, component = c("endemic", "epidemic"),
    trace = 2, ...)
## S3 method for class 'twinstim'
drop1(object, scope, component = c("endemic", "epidemic"),
     trace = 2, ...)
```

Arguments

`object`	an object of class `"twinstim"`.
`component`	one of `"endemic"` or `"epidemic"` (partially matched), determining the model component where the algorithm should proceed.
`scope,direction,trace`	see `step` and `add1`, respectively.
`verbose`	see `twinstim`.
`...`	further arguments passed to `step`, `add1.default`, or `drop1.default`, respectively.

Value

See step and add1, respectively.

Author(s)

(of this wrapper around step) Sebastian Meyer

See Also

step, add1, drop1

Examples

```
data("imdepi")
data("imdepifit")

## simple baseline model
m0 <- update(imdepifit, epidemic=~1, siaf=NULL, start=c("e.(Intercept)"=-17))

## AIC-based step-wise backward selection of the endemic component
m0_step <- stepComponent(m0, "endemic", scope=list(lower=~I(start/365-3.5)))
## nothing is dropped from the model
```

---

twinstim_tiaf *Temporal Interaction Function Objects*

Description

A temporal interaction function for use in twinstim can be constructed via the tiaf function. It checks the supplied function elements, assigns defaults for missing arguments, and returns all checked arguments in a list. However, for standard applications it is much easier to use one of the pre-defined temporal interaction functions, e.g., tiaf.exponential.

Usage

```
tiaf(g, G, deriv, Deriv, npars, validpars = NULL)
```

Arguments

g	the temporal interaction function. It must accept two arguments, the first one being a vector of time points, the second one a parameter vector. For marked twinstim, it must accept the type of the event (integer code) as its third argument (either a single type for all locations or separate types for each location).
G	a primitive of $g(t)$ (with respect to time). It must accept the same arguments as g, for instance a *vector* of time points (not just a single one).

`deriv` optional derivative of $g(t)$ *with respect to the parameters*. It takes the same arguments as g but returns a matrix with as many rows as there were time points in the input and `npars` columns. This derivative is necessary for the calculation of the score function in `twinstim()`, which is advantageous for the numerical log-likelihood maximization.

`Deriv` optional primitive of `deriv` (with respect to time). It must accept the same arguments as `deriv`, `g` and `G` and returns a matrix with as many rows as there were time points in the input and `npars` columns. The integrated derivative is necessary for the score function in `twinstim`.

`npars` the number of parameters of the temporal interaction function g (i.e. the length of its second argument).

`validpars` optional function taking one argument, the parameter vector, indicating if it is valid. This approach to specify parameter constraints is rarely needed, because usual box-constrained parameters can be taken into account by using L-BFGS-B as the optimization method in `twinstim` (with arguments `lower` and `upper`), and positivity constraints by using log-parametrizations. This component is not necessary (and ignored) if `npars == 0`.

Value

list of checked arguments.

Author(s)

Sebastian Meyer

See Also

`tiaf.exponential` for a pre-defined temporal interaction function, and `siaf` for the spatial interaction function.

---

twinstim_update update-*method for* "twinstim"

Description

Update and (by default) re-fit a "twinstim". This method is especially useful if one wants to add the `model` environment (which is required for some methods) to a fitted model object a posteriori.

Usage

```
## S3 method for class 'twinstim'
update(object, endemic, epidemic,
       control.siaf, optim.args, model,
       ..., use.estimates = TRUE, evaluate = TRUE)
```

Arguments

object	a previous "twinstim" fit.
endemic, epidemic	changes to the formulae – see update.formula and twinstim.
control.siaf	a list (see twinstim) to replace the given elements in the original control.siaf list. If NULL, the original list of control arguments is removed from the call, i.e., the defaults are used in twinstim.
optim.args	see twinstim. If a list, it will modify the original optim.args using modifyList.
model	see twinstim. If this is the only argument to update, re-fitting is cleverly circumvented. Enriching the fit by the model environment is, e.g., required for intensityplot.twinstim.
...	Additional arguments to the call, or arguments with changed values. If start values are specified, they need to be in the same format as in the original call object$call$start, which is either a named list of named numeric vectors or a named numeric vector; see the argument description in twinstim.
use.estimates	logical indicating if the estimates of object should be used as initial values for the new fit (in the start argument of twinstim). Defaults to TRUE.
evaluate	If TRUE (default), evaluate the new call else return the call.

Value

If evaluate = TRUE the re-fitted object, otherwise the updated call.

Author(s)

Sebastian Meyer

Inspiration and some pieces of code originate from update.default by the R Core Team.

See Also

update.default

Examples

```
data("imdepi")
data("imdepifit")

## add another epidemic covariate (but fix siaf-parameter for speed)
imdepifit2 <- update(imdepifit, epidemic = ~. + log(popdensity),
                     optim.args = list(fixed="e.siaf.1"))

## compare by AIC
AIC(imdepifit, imdepifit2)
```

unionSpatialPolygons *Compute the Unary Union of* "SpatialPolygons"

Description

Union all subpolygons of a "SpatialPolygons" object. This is a wrapper for the polygon clipping engines implemented by packages **rgeos**, **polyclip**, or **gpclib**.

Usage

```
unionSpatialPolygons(SpP, method = c("rgeos", "polyclip", "gpclib"), ...)
```

Arguments

SpP	an object of class "SpatialPolygons". For the **polyclip** method only, all polygon classes for which an xylist-method exists should work as input.
method	polygon clipping machinery to use. Default is to simply call gUnaryUnion in package **rgeos**. For method="polyclip", function polyclip from package **polyclip** is used, whereas method="gpclib" calls unionSpatialPolygons in package **maptools** (and requires acceptance of **gpclib**'s restricted license via surveillance.options(gpclib=TRUE)).
...	further arguments passed to the chosen method.

Value

an object of class "SpatialPolygons" representing the union of all subpolygons.

Author(s)

Sebastian Meyer

See Also

gUnaryUnion in package **rgeos**, polyclip in package **polyclip**, unionSpatialPolygons in package **maptools** (for using union of package **gpclib**).

Examples

```
## Load districts of Germany
load(system.file("shapes", "districtsD.RData", package="surveillance"))

## Union these districts
plot(unionSpatialPolygons(districtsD, method="polyclip"), border=0, col=1)
plot(districtsD, add=TRUE, border="lightgrey")
if (requireNamespace("rgeos"))
    plot(unionSpatialPolygons(districtsD, method="rgeos"),
         add=TRUE, border=2, lwd=2)
```

untie *Randomly Break Ties in Data*

Description

This is a generic function intended to randomly break tied data in a way similar to what jitter does: tie-breaking is performed by shifting *all* data points by a random amount. The **surveillance** package defines methods for matrices, "epidataCS", and a default method for numeric vectors.

Usage

```
untie(x, amount, ...)

## S3 method for class 'epidataCS'
untie(x, amount = list(t=NULL, s=NULL),
      minsep = list(t=0, s=0), direction = "left", keep.sources = FALSE,
      ..., verbose = FALSE)
## S3 method for class 'matrix'
untie(x, amount = NULL, minsep = 0,
      constraint = NULL, giveup = 1000, ...)
## Default S3 method:
untie(x, amount = NULL, minsep = 0,
      direction = c("symmetric", "left", "right"), sort = NULL,
      giveup = 1000, ...)
```

Arguments

x	the data to be untied.
amount	upperbound for the random amount by which data are shifted. NULL means to use a data-driven default, which equals the minimum separation of the data points for the non-symmetric default method and its half for the symmetric default method and the matrix method. For numeric vectors (default method), the jittered version is the same as for jitter(x, amount=amount) if direction="symmetric" (and amount is non-NULL), and x "+-" runif(length(x), 0, amount) (otherwise). For matrices, a vector uniformly drawn from the disc with radius amount is added to each point (row). For "epidataCS", amount is a list stating the amounts for the temporal and/or spatial dimension, respectively. It then uses the specific methods with arguments constraint=x$W, direction, and sort=TRUE.
minsep	minimum separation of jittered points. Can only be obeyed if much smaller than amount (also depending on the number of points). minsep>0 is currently only implemented for the spatial (matrix) method.
keep.sources	logical (FALSE). If TRUE, the original list of possible event sources in x$events$.sources will be preserved. For instance, events observed at the same time did by definition not trigger each other; however, after random tie-breaking one event will

	precede the other and considered as a potential source of infection for the latter, although it could just as well be the other way round. Enabling `keep.sources` will use the `.sources` list from the original (tied) `"epidataCS"` object.
`constraint`	an object of class `"SpatialPolygons"` representing the domain which the points of the matrix should belong to – before and after jittering.
`giveup`	number of attempts after which the algorithm should stop trying to generate new points.
`direction`	one of `"symmetric"` (default), `"left"`, or `"right"`, indicating in which direction vector elements should be shifted.
`sort`	logical indicating if the jittered vector should be sorted. Defaults to doing so if the original vector was already sorted.
`...`	For the `"epidataCS"`-method: arguments passed to the `matrix`- or `default`-method (`giveup`). Unused in other methods.
`verbose`	logical passed to `as.epidataCS`.

Value

the untied (jittered) data.

Author(s)

Sebastian Meyer

See Also

`jitter`

Examples

```
# vector example
set.seed(123)
untie(c(rep(1,3), rep(1.2, 4), rep(3,3)), direction="left", sort=FALSE)

# spatial example
data(imdepi)
coords <- coordinates(imdepi$events)
table(duplicated(coords))
plot(coords, cex=sqrt(multiplicity(coords)))
set.seed(1)
coords_untied <- untie(coords)
stopifnot(!anyDuplicated(coords_untied))
points(coords_untied, col=2) # shifted by very small amount in this case
```

wrap.algo *Multivariate Surveillance through independent univariate algorithms*

Description

This function takes an sts object and applies an univariate surveillance algorithm to the time series of each observational unit.

Usage

```
wrap.algo(sts, algo, control,control.hook=function(k)
         return(control),verbose=TRUE,...)

farrington(sts, control=list(range=NULL, b=3, w=3,reweight=TRUE,
         verbose=FALSE,alpha=0.01),...)
bayes(sts, control = list(range = range, b = 0, w = 6,
         actY = TRUE,alpha=0.05),...)
rki(sts, control = list(range = range, b = 2, w = 4,
         actY = FALSE),...)
cusum(sts,  control = list(range=range, k=1.04, h=2.26,
         m=NULL, trans="standard",alpha=NULL),...)
glrpois(sts, control = list(range=range,c.ARL=5, S=1,beta=NULL,
         Mtilde=1, M=-1, change="intercept",theta=NULL),...)
glrnb(sts, control = list(range=range,c.ARL=5, mu0=NULL, alpha=0,
         Mtilde=1, M=-1, change="intercept",
         theta=NULL,dir=c("inc","dec"),
         ret=c("cases","value")),...)
outbreakP(sts, control=list(range = range, k=100,
         ret=c("cases","value"),maxUpperboundCases=1e5),...)
```

Arguments

sts	Object of class sts
algo	Character string giving the function name of the algorithm to call, e.g. "algo.farrington". Calling is done using do.call.
control	Control object as list. Depends on each algorithm.
control.hook	This is a function for handling multivariate objects. This argument is a function function of integer k, which returns the appropriate control object for region k
verbose	Boolean, if TRUE then textual information about the process is given
...	Additional arguments sent to the algo function.

Value

An sts object with the alarm, upperbound, etc. slots filled with the results of independent and univariate surveillance algorithm.

Author(s)

M. Höhle

See Also

algo.rki, algo.farrington, algo.cusum, algo.glrpois, algo.glrnb, algo.outbreakP for the exact form of the control object.

---

xtable.algoQV *Xtable quality value object*

Description

Xtable a single qualitity value object in a nicely formatted way

Usage

```
  ## S3 method for class 'algoQV'
xtable(x,caption = NULL, label = NULL,
            align = NULL, digits = NULL, display = NULL, ...)
```

Arguments

x	Quality Values object generated with quality
caption	See xtable
label	See xtable
align	See xtable
digits	See xtable
display	See xtable
...	Further arguments (see xtable)

See Also

xtable

Examples

```
# Create a test object
disProgObj <- sim.pointSource(p = 0.99, r = 0.5, length = 200, A = 1,
                                    alpha = 1, beta = 0, phi = 0,
                                    frequency = 1, state = NULL, K = 1.7)

# Let this object be tested from rki1
survResObj <- algo.rki1(disProgObj, control = list(range = 50:200))

# Compute the quality values in a nice formatted way
xtable(algo.quality(survResObj))
```

`zetaweights` *Power-Law Weights According to Neighbourhood Order*

Description

Compute power-law weights with decay parameter `d` based on a matrix of neighbourhood orders `nbmat` (e.g., as obtained via `nbOrder`). Without normalization and truncation, this is just o^{-d} (where o is a neighbourhood order). This function is mainly used internally for `W_powerlaw` weights in `hhh4` models.

Usage

```
zetaweights(nbmat, d = 1, maxlag = max(nbmat), normalize = FALSE)
```

Arguments

`nbmat`	numeric, symmetric matrix of neighbourhood orders.
`d`	single numeric decay parameter (default: 1). Should be positive.
`maxlag`	single numeric specifying an upper limit for the power law. For neighbourhood orders > `maxlag`, the resulting weight is 0. Defaults to no truncation.
`normalize`	Should the resulting weight matrix be normalized such that rows sum to 1?

Value

a numeric matrix with same dimensions and names as the input matrix.

Author(s)

Sebastian Meyer

See Also

`W_powerlaw`

Examples

```
nbmat <- matrix(c(0,1,2,2,
                  1,0,1,1,
                  2,1,0,2,
                  2,1,2,0), 4, 4, byrow=TRUE)
zetaweights(nbmat, d=1, normalize=FALSE) # harmonic: o^-1
zetaweights(nbmat, d=1, normalize=TRUE)  # rowSums=1
zetaweights(nbmat, maxlag=1, normalize=FALSE) # results in adjacency matrix
```

`[,sts-methods` *Extraction and Subsetting of* **sts** *objects*

Description

Methods for `"["`, i.e., extraction or subsetting of the `"sts"` class in package **surveillance**.

Note that `[<--methods` methods (i.e. subassigments) are currently not supported.

`drop` is always `FALSE`.

Methods

There are more than these:

x = "sts", i = "missing", j = "missing", drop= "ANY" ...

x = "sts", i = "numeric", j = "missing", drop= "missing" ...

x = "sts", i = "missing", j = "numeric", drop= "missing" ...

Examples

```
data("ha.sts")
haagg <- aggregate(ha.sts, nfreq=13)

#A suite of of simple tests (inspired by the Matrix package)
#stopifnot(identical(ha.sts, ha.sts[]))


plot(haagg[, 3])       # Single series
plot(haagg[1:30, 3])   # Somewhat shorter

#Counts at time 20
plot(haagg[20, ], type = observed ~1 |unit)
```

Index