Package: surveillance (via r-universe)

October 19, 2026

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

Version 1.26.1.9000

Date 2026-09-07

Depends R (>= 3.6.0), methods, grDevices, graphics, stats, utils, sp (>= 2.1-4)

Imports polyCub (>= 0.8.0), MASS, Matrix, nlme, spatstat.geom

LinkingTo polyCub

Suggests parallel, grid, xtable (>= 1.7-0), gridExtra (>= 2.0.0), lattice (>= 0.20-44), scales, animation, msm, spc, coda, runjags, INLA, spdep, numDeriv, maxLik, gsl, fanplot, quadprog, memoise, polyclip, intervals, splancs, gamlss, MGLM (>= 0.1.0), sf, tinytest (>= 1.4.1), knitr, bibtex

Enhances xts, ggplot2

Description Statistical methods for the modeling and monitoring of time series of counts, proportions and categorical data, as well as for the modeling of continuous-time point processes of epidemic phenomena. The monitoring methods focus on aberration detection in count data time series from public health surveillance of communicable diseases, but applications could just as well originate from environmetrics, reliability engineering, econometrics, or social sciences. The package implements many typical outbreak detection procedures such as the (improved) Farrington algorithm, or the negative binomial GLR-CUSUM method of Hoehle and Paul (2008) <doi:10.1016/j.csda.2008.02.015>. A novel CUSUM approach combining logistic and multinomial logistic modeling is also included. The package contains several real-world data sets, the ability to simulate outbreak data, and to visualize the results of the monitoring in a temporal, spatial or spatio-temporal fashion. A recent overview of the available monitoring procedures is given by Salmon et al. (2016) <doi:10.18637/jss.v070.i10>. For the retrospective analysis of epidemic spread, the package provides three endemic-epidemic

modeling frameworks with tools for visualization, likelihood inference, and simulation. hhh4() estimates models for (multivariate) count time series following Paul and Held (2011) <doi:10.1002/sim.4177> and Meyer and Held (2014) <doi:10.1214/14-AOAS743>. twinSIR() models the susceptible-infectious-recovered (SIR) event history of a fixed population, e.g, epidemics across farms or networks, as a multivariate point process as proposed by Hoehle (2009) <doi:10.1002/bimj.200900050>. twinstim() estimates self-exciting point process models for a spatio-temporal point pattern of infective events, e.g., time-stamped geo-referenced surveillance data, as proposed by Meyer et al. (2012) <doi:10.1111/j.1541-0420.2011.01684.x>. A recent overview of the implemented space-time modeling frameworks for epidemic phenomena is given by Meyer et al. (2017) <doi:10.18637/jss.v077.i11>.

License GPL-2

URL https://EE-hub.codeberg.page/surveillance/

BugReports https://codeberg.org/EE-hub/surveillance/issues

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

VignetteBuilder utils, knitr

Note Building the package requires R >= 4.6.0 for Rd \bibshow{} et al.

Repository https://ee-lib.r-universe.dev

Date/Publication 2026-10-19 13:29:16 UTC

RemoteUrl https://codeberg.org/EE-hub/surveillance

RemoteRef HEAD

RemoteSha ac9caa637af7231d0bd6b9b1525f1fa5383e20e6

Contents

surveillance-package . . . 6
abattoir . . . 8
addFormattedXAxis . . . 9
addSeason2formula . . . 10
aggregate-methods . . . 11
algo.bayes . . . 12
algo.call . . . 14
algo.cdc . . . 15
algo.compare . . . 17
algo.cusum . . . 18
algo.farrington . . . 20
algo.farrington.assign.weights . . . 23
algo.farrington.fitGLM . . . 23

algo.farrington.threshold . . . 25
algo.glrnb . . . 25
algo.hmm . . . 29
algo.outbreakP . . . 32
algo.quality . . . 34
algo.rki . . . 35
algo.rogerson . . . 37
algo.summary . . . 39
all.equal . . . 40
animate . . . 41
anscombe.residuals . . . 41
arlCusum . . . 42
backprojNP . . . 43
bestCombination . . . 47
boda . . . 48
bodaDelay . . . 50
calibrationTest . . . 53
campyDE . . . 54
categoricalCUSUM . . . 56
checkResidualProcess . . . 59
clapply . . . 60
coeflist . . . 61
deleval . . . 62
discpoly . . . 63
disProg2sts . . . 64
earsC . . . 65
epidata . . . 68
epidata_animate . . . 73
epidata_intersperse . . . 76
epidata_plot . . . 77
epidata_summary . . . 79
epidataCS . . . 81
epidataCS_aggregate . . . 87
epidataCS_animate . . . 89
epidataCS_permute . . . 92
epidataCS_plot . . . 93
epidataCS_update . . . 96
fanplot . . . 97
farringtonFlexible . . . 99
find.kh . . . 103
findH . . . 104
findK . . . 105
fluBYBW . . . 106
formatDate . . . 107
formatPval . . . 108
glm_epidataCS . . . 108
ha . . . 110
hagelloch . . . 110

hepatitisA . . . 113
hhh4 . . . 114
hhh4_formula . . . 123
hhh4_methods . . . 124
hhh4_plot . . . 127
hhh4_predict . . . 133
hhh4_simulate . . . 135
hhh4_simulate_plot . . . 138
hhh4_simulate_scores . . . 141
hhh4_update . . . 143
hhh4_validation . . . 145
hhh4_W . . . 150
hhh4_W_utils . . . 153
husO104Hosp . . . 154
imdepi . . . 155
imdepifit . . . 158
influMen . . . 159
intensityplot . . . 159
intersectPolyCircle . . . 160
isoWeekYear . . . 161
knox . . . 162
ks.plot.unif . . . 164
layout.labels . . . 166
linelist2sts . . . 167
LRCUSUM.runlength . . . 169
m1 . . . 172
magic.dim . . . 173
makeControl . . . 174
marks . . . 174
measles.weser . . . 175
measlesDE . . . 176
meningo.age . . . 177
MMRcoverageDE . . . 178
momo . . . 179
multiplicity . . . 180
multiplicity.Spatial . . . 180
nbOrder . . . 181
nowcast . . . 182
pairedbinCUSUM . . . 187
permutationTest . . . 190
pit . . . 192
plapply . . . 193
poly2adjmat . . . 195
polyAtBorder . . . 196
primeFactors . . . 197
R0 . . . 198
ranef . . . 201
refvalIdxByDate . . . 201

residualsCT . . . 202
rotaBB . . . 203
salmAllOnset . . . 203
salmHospitalized . . . 204
salmNewport . . . 204
salmonella.agona . . . 205
scores . . . 205
shadar . . . 207
sim.pointSource . . . 208
sim.seasonalNoise . . . 209
stcd . . . 210
stK . . . 212
sts-class . . . 214
sts_animate . . . 218
sts_conversion . . . 220
sts_creation . . . 221
sts_ggplot . . . 222
sts_observation . . . 224
stsBP-class . . . 224
stsNC-class . . . 225
stsNClist_animate . . . 226
stsNewport . . . 227
stsplot . . . 227
stsplot_space . . . 228
stsplot_time . . . 231
stsSlot-generics . . . 234
stsXtrct . . . 234
surveillance.options . . . 236
tidy.sts . . . 237
toLatex.sts . . . 238
twinSIR . . . 240
twinSIR_cox . . . 245
twinSIR_intensityplot . . . 245
twinSIR_methods . . . 248
twinSIR_profile . . . 250
twinSIR_simulation . . . 252
twinstim . . . 257
twinstim_epitest . . . 265
twinstim_iaf . . . 268
twinstim_iafplot . . . 273
twinstim_intensity . . . 276
twinstim_methods . . . 279
twinstim_plot . . . 282
twinstim_profile . . . 283
twinstim_siaf . . . 284
twinstim_simEndemicEvents . . . 286
twinstim_simulation . . . 287
twinstim_step . . . 293

twinstim_tiaf 295
twinstim_update 296
unionSpatialPolygons 297
untie 298
wrap.algo 300
zetaweights 301

Index **303**

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

Description

The R package **surveillance** implements statistical methods for the retrospective modeling and prospective monitoring of epidemic phenomena in temporal and spatio-temporal contexts. Focus is on (routinely collected) public health surveillance data, but the methods just as well apply to data from environmetrics, econometrics or the social sciences. As many of the monitoring methods rely on statistical process control methodology, the package is also relevant to quality control and reliability engineering.

Details

The package implements many 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), see `wrap.algo`. Also CUSUMs for the prospective change-point detection in binomial, beta-binomial and multinomial time series are covered based on generalized linear modeling, see `categoricalCUSUM`. This includes, e.g., paired comparison Bradley-Terry modeling described in Höhle (2010), or paired binary CUSUM (`pairedbinCUSUM`) described by Steiner et al. (1999). 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. In dealing with time series data, the fundamental data structure of the package is the S4 class `sts` wrapping observations, monitoring results and date handling for multivariate time series. A recent overview of the available monitoring procedures is given by Salmon, Schumacher, and Höhle (2016).

For the retrospective analysis of epidemic spread, the package provides three endemic-epidemic modeling frameworks with tools for visualization, likelihood inference, and simulation. The function `hhh4` offers inference methods for the (multivariate) count time series model of Held et al. (2005), including some of its extensions (Paul and Held, 2011; Meyer and Held, 2014). See `vignette("hhh4")` for a general introduction and `vignette("hhh4_spacetime")` for a discussion and illustration of spatial hhh4 models. Self-exciting point processes are modeled through endemic-epidemic conditional intensity functions. `twinSIR` (Höhle, 2009) models the susceptible-infectious-recovered (SIR) event history of a fixed population, e.g, epidemics across farms or networks; see `vignette("twinSIR")` for an illustration. `twinstim` (Meyer et al., 2012) fits spatio-temporal point process models to point patterns of infective events, e.g., time-stamped geo-referenced surveillance data on infectious disease occurrence; see `vignette("twinstim")` for an illustration. A recent

overview of the implemented space-time modeling frameworks for epidemic phenomena is given by Meyer, Held, and Höhle (2017).

'CITATION'

To cite the R package **surveillance** in publications, please use:

For the monitoring functions: Salmon M, Schumacher D, Höhle M (2016). "Monitoring count time series in R: Aberration detection in public health surveillance." *Journal of Statistical Software*, **70**(10), 1–35. doi:10.18637/jss.v070.i10.

For the modeling frameworks: Meyer S, Held L, Höhle M (2017). "Spatio-temporal analysis of epidemic phenomena using the R package **surveillance**." *Journal of Statistical Software*, **77**(11), 1–55. doi:10.18637/jss.v077.i11.

See also `citation("surveillance")`.

Author(s)

- Michael Hoehle (ORCID: 0000-0002-0423-6702)
- Sebastian Meyer (ORCID: 0000-0002-1791-9449)
- Michaela Paul

Maintainer: Sebastian Meyer <seb.meyer@fau.de>

The authors acknowledge code contributions by Leonhard Held, Howard Burkom, Thais Correa, Mathias Hofmann, Christian Lang, Juliane Manitz, Sophie Reichert, Andrea Riebler, Daniel Sabanes Bove, Maelle Salmon, Dirk Schumacher, Stefan Steiner, Mikko Virtanen, Wei Wei, Valentin Wimmer.

Furthermore, the authors would like to thank the following people for ideas, discussions, testing and feedback: Doris Altmann, Johannes Bracher, Caterina De Bacco, Johannes Dreesman, Johannes Elias, Marc Geilhufe, Jim Hester, Kurt Hornik, Mayeul Kauffmann, Junyi Lu, Lore Merdrignac, Tim Pollington, Marcos Prates, André Victor Ribeiro Amaral, Brian D. Ripley, François Rousseu, Barry Rowlingson, Christopher W. Ryan, Klaus Stark, Yann Le Strat, André Michael Toschke, Wei Wei, George Wood, Achim Zeileis, Bing Zhang.

References

References to the underlying methodological papers are given on the specific help pages; some are also available in an installed BibTeX file:

```
system.file("REFERENCES.bib", package = "surveillance")
```

See Also

Website: https://EE-hub.codeberg.page/surveillance/

Examples

```
## Additional documentation and illustrations of the methods are
## available in the form of package vignettes and demo scripts:
vignette(package = "surveillance")
demo(package = "surveillance")
```

---

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): Online change-point detection in categorical time series. In: T. Kneib and G. Tutz (Eds.), Statistical Modelling and Regression Structures, Physica-Verlag.

See Also

categoricalCUSUM

Examples

```
data("abattoir")
plot(abattoir)
population(abattoir)
```

addFormattedXAxis *Formatted Time Axis for "sts" Objects*

Description

Add a nicely formatted x-axis to time series plots related to the "sts" class. This utility function is used by, e.g., stsplot_time1 and plotHHH4_fitted1.

Usage

```
addFormattedXAxis(x, epochsAsDate = FALSE,
                  xaxis.tickFreq = list("%Q"=atChange),
                  xaxis.labelFreq = xaxis.tickFreq,
                  xaxis.labelFormat = "%G\n\n%OQ",
                  ...)

## helper functions for time axis formatting
atChange(x, xm1)
at2ndChange(x, xm1)
atMedian(x, xm1)
```

Arguments

x, xm1	for addFormattedXAxis, x is an object of class "sts". For the helper functions, x and xm1 are numeric vectors, see the details below.
epochsAsDate	a logical indicating if the old (FALSE) or the new (TRUE) and more flexible implementation should be used. The xaxis.* arguments are only relevant for the new implementation epochsAsDate = TRUE.
xaxis.labelFormat, xaxis.tickFreq, xaxis.labelFreq	see the details below.
...	further arguments passed to axis.

Details

The setting epochsAsDate = TRUE enables very flexible formatting of the x-axis and its annotations using the xaxis.tickFreq, xaxis.labelFreq and xaxis.labelFormat arguments. 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. Please note that the input to the subsetting function is converted using as.numeric before calling the function. Hence, the conversion specification needs to result in a string convertible to integer.

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)
    as.integer(tapply(seq_along(x), x, quantile, probs=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 function is called for its side effects.

Author(s)

Michael Höhle with contributions by Sebastian Meyer

See Also

the examples in `stsplot_time1` and `plotHHH4_fitted1`

`addSeason2formula` *Add Harmonics 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 using the special `fe` syntax supported by `hhh4`.
`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

$$\sin(s \cdot 2\pi \cdot \texttt{timevar}/\texttt{period}),\ \cos(s \cdot 2\pi \cdot \texttt{timevar}/\texttt{period}),$$

for $s = 1, \ldots, \texttt{S}$ to an existing formula `f`.

Note the following equivalence when interpreting the coefficients of the seasonal terms:

$$\gamma \sin(\omega t) + \delta \cos(\omega t) = A \sin(\omega t + \epsilon)$$

with amplitude $A = \sqrt{\gamma^2 + \delta^2}$ and phase shift $\epsilon = \arctan(\delta/\gamma)$. The amplitude and phase shift can be obtained from a fitted `hhh4` model via `coef(..., amplitudeShift = TRUE)`, see `coef.hhh4`.

Value

Returns a `formula` with the seasonal terms added.

Note

To use the resulting formula in `hhh4`, a time variable named as specified by the argument `timevar` must be available.

In **surveillance** $\leq$ 1.25.0, the formula environment was always set to `.GlobalEnv`; it is now preserved from the input.

Author(s)

M. Paul, with contributions by S. Meyer

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 an "sts" Object Over Time or Across Units*

Description

Aggregate the matrix slots of an `"sts"` object. Either the time series is aggregated so a new sampling frequency of `nfreq` observations per year is obtained (i.e., as in `aggregate.ts`), or the aggregation is over all columns (units).

Usage

```
## S4 method for signature 'sts'
aggregate(x, by = "time", nfreq = "all", ...)
```

Arguments

x	an object of class "sts".
by	a string being either "time" or "unit".
nfreq	new sampling frequency for by="time". If nfreq="all" then all time points are summed.
...	unused (argument of the generic).

Value

an object of class "sts".

Warning

Aggregation over units fills the upperbound slot with NAs and the map slot is left as-is, but the object cannot be plotted by unit any longer.

The populationFrac slot is aggregated just like observed. Population fractions are recomputed if and only if x is no multinomialTS and already contains population fractions. This might not be intended, especially for aggregation over time.

Examples

```
data("ha.sts")
dim(ha.sts)
dim(aggregate(ha.sts, by = "unit"))
dim(aggregate(ha.sts, nfreq = 13))
```

---

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 should 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 timepoints to be evaluated by the actual system.

Value

a 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",
                              range = range, b = 1, w = 5, actY = TRUE,alpha=0.05)
                     ))
# show selected survRes objects
names(survRes)
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 should 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

`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

References

Stroup, D., G. Williamson, J. Herndon, and J. Karon (1989). Detection of aberrations in the occurrence of notifiable diseases surveillance data. *Statistics in Medicine* 8, 323–329. doi:10.1002/sim.4780080312

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 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",
                             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, reset = FALSE))
```

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 asymptotic normality) - This is the default.
- `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 $m + \alpha * m^2$

`reset` logical: Should the CUSUM statistic be reset to 0 immediately after an alarm? This is the traditional form of the chart as used in industrial process control, but not the default choice in outbreak detection when continuous periods of abnormal disease activity should be flagged.

Value

`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, it is *not* `reset` by default, i.e., signals occur until the CUSUM statistic again returns below the threshold.

In case `m="glm"` was used, the returned `control$m.glm` entry contains the fitted `"glm"` object.

Note

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

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
set.seed(321)
stsObj <- sts(observed = rpois(500,lambda=5))
# there should be no alarms as mean doesn't change
res <- cusum(stsObj, control = list(range = 100:500, trans = "anscombe"))
plot(res, xaxis.labelFormat = NULL)

# 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 weeks 200 to 250 for outbreaks
surv0 <- algo.cusum(disProgObj, control = list(range = 200:250))
plot(surv0, xaxis.years = FALSE)

# alternatively, using the newer "sts" interface
stsObj <- disProg2sts(disProgObj)
surv <- cusum(stsObj, control = list(range = 200:250))
plot(surv)
stopifnot(upperbound(surv) == surv0$upperbound)
```

`algo.farrington` *Surveillance for Count Time Series Using the Classic Farrington Method*

Description

Implements the procedure of Farrington et al. (1996). At each time point of the specified `range`, a GLM is fitted to predict the counts. This is then compared to the observed counts. If the observation is above a specific quantile of the prediction interval, then an alarm is raised.

Usage

```
# original interface for a single "disProg" time series
algo.farrington(disProgObj, control=list(
    range=NULL, b=5, w=3, reweight=TRUE, verbose=FALSE, plot=FALSE,
    alpha=0.05, trend=TRUE, limit54=c(5,4), powertrans="2/3",
    fitFun="algo.farrington.fitGLM.fast"))

# wrapper for "sts" data, possibly multivariate
farrington(sts, control=list(
    range=NULL, b=5, w=3, reweight=TRUE, verbose=FALSE,
    alpha=0.05), ...)
```

Arguments

`disProgObj` an object of class "`disProg`" (a list including `observed` and `state` time series).

`control` list of control parameters

- `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.

`sts` an object of class "`sts`".

`...` arguments for `wrap.algo`, e.g., `verbose=FALSE`.

Details

The following steps are performed 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

For `algo.farrington`, a list object of class `"survRes"` with elements `alarm`, `upperbound`, `trend`, `disProgObj`, and `control`.

For `farrington`, the input `"sts"` object with updated `alarm`, `upperbound` and `control` slots, and subsetted to `control$range`.

Author(s)

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.

See Also

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

An improved Farrington algorithm is available as function `farringtonFlexible`.

Examples

```
#load "disProg" data
data("salmonella.agona")

#Do surveillance for the last 42 weeks
n <- length(salmonella.agona$observed)
control <- list(b=4,w=3,range=(n-42):n,reweight=TRUE, verbose=FALSE,alpha=0.01)
res <- algo.farrington(salmonella.agona,control=control)
plot(res)

#Generate Poisson counts and create an "sts" object
set.seed(123)
x <- rpois(520,lambda=1)
stsObj <- sts(observed=x, frequency=52)

if (surveillance.options("allExamples")) {
#Compare timing of the two possible fitters for algo.farrington
  range <- 312:520
  system.time( sts1 <- farrington(stsObj, control=list(range=range,
                        fitFun="algo.farrington.fitGLM.fast"), verbose=FALSE))
  system.time( sts2 <- farrington(stsObj, control=list(range=range,
                        fitFun="algo.farrington.fitGLM"), verbose=FALSE))
  #Check if results are the same
```

```
  stopifnot(upperbound(sts1) == upperbound(sts2))
}
```

---

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

anscombe.residuals

---

algo.farrington.fitGLM
Fit 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 (superfluous?)
`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 as proposed by Höhle and Paul (2008). The implementation is described in Salmon et al. (2016).

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"), xMax=1e4))

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"), xMax=1e4))
```

Arguments

`disProgObj` object of class `disProg` to do surveillance for. For new `sts`-class data, use the `glrnb` wrapper, or the `sts2disProg` converter.

`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 Poisson / negative binomial distribution with the same length as `range`. If `NULL` the observed values in `1:(min(range)-1)` are used to estimate the beta vector through a generalized linear model. To fine-tune the model one can instead specify `mu0` as a list with two components:
  - `S` integer number of harmonics to include (typically 1 or 2)
  - `trend` A Boolean indicating whether to include a term `t` in the GLM model

  The fitting is controlled by the `estimateGLRNbHook` function. The in-control mean model is re-fitted after every alarm. The fitted models can be found as a list `mod` in the `control` slot after the call.
  Note: If a value for `alpha` is given, then the inverse of this value is used as fixed `theta` in a `negative.binomial` `glm`. If `is.null(alpha)` then the parameter is estimated as well (using `glm.nb`) – see the description of this parameter for details.
- `alpha` The (known) dispersion parameter of the negative binomial distribution, i.e. the parametrization of the negative binomial is such that the variance is $mean + alpha * mean^2$. Note: This parametrization is the inverse of the shape parametrization used in R – for example in `dnbinom` and `glr.nb`. Hence, 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. However, the parameter is estimated only once from the first fit. Subsequent fittings are only for the parameters of the linear predictor with `alpha` fixed.
- `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` (default), the *GLR scheme* is used (with direction `dir`). If not `NULL` and `change="intercept"`, the given `theta` $= \kappa > 0$ is used in a *recursive LR scheme*, which is faster (but always assumes an *inc*rease).

`dir` a string specifying the direction of testing in the *GLR scheme*. With `"inc"` (default), 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).

`xMax` Maximum value to try for x to see if this is the upperbound number of cases before sounding an alarm (Default: 1e4). This only applies for the GLR using the NegBin when `ret="cases"` – see details.

Details

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

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

where instead of $1 \le k \le n$ the GLR statistic is computed for all $k \in \{n - M, \ldots, n - \tilde{M} + 1\}$. To achieve the typical behaviour from $1 \le k \le n$ 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 reset 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.

In case `is.null(theta)` and `alpha>0` as well as `ret="cases"` then a brute-force search is conducted for each time point in range in order to determine the number of cases necessary before an alarm is sounded. In case no alarm was sounded so far by time t, the function increases $x[t]$ until an alarm is sounded any time before time point t. If no alarm is sounded by `xMax`, a return value of 1e99 is given. Similarly, if an alarm was sounded by time t the function counts down instead. Note: This is slow experimental code!

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

`algo.glrpois` simply calls `algo.glrnb` with `control$alpha` set to 0.

`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 time point $\leq n$. Both lead to the same alarm timepoints, but `"cases"` has an obvious interpretation.

Author(s)

M. Höhle with contributions by V. Wimmer

References

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

Salmon, M., Schumacher, D. and Höhle, M. (2016): Monitoring count time series in R: Aberration detection in public health surveillance. *Journal of Statistical Software*, **70** (10), 1-35. doi:10.18637/jss.v070.i10

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)


## Poisson example
#Generate data
set.seed(42)
x <- rpois(length(t),mu0*(exp(kappa)^(t>=tau)))
s.ts <- sts(observed=x, state=(t>=tau))
#Plot the data
plot(s.ts, xaxis.labelFormat=NULL)
#Run
cntrl = list(range=t,c.ARL=5, Mtilde=1, mu0=mu0,
             change="intercept",ret="value",dir="inc")
glr.ts <- glrpois(s.ts,control=cntrl)
plot(glr.ts, xaxis.labelFormat=NULL, dx.upperbound=0.5)
lr.ts  <- glrpois(s.ts,control=c(cntrl,theta=0.4))
plot(lr.ts, xaxis.labelFormat=NULL, dx.upperbound=0.5)

#using the legacy interface for "disProg" data
lr.ts0  <- algo.glrpois(sts2disProg(s.ts), control=c(cntrl,theta=0.4))
stopifnot(upperbound(lr.ts) == lr.ts0$upperbound)
```

```
## NegBin example
#Generate data
set.seed(42)
alpha <- 0.2
x <- rnbinom(length(t),mu=mu0*(exp(kappa)^(t>=tau)),size=1/alpha)
s.ts <- sts(observed=x, state=(t>=tau))

#Plot the data
plot(s.ts, xaxis.labelFormat=NULL)

#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 <- glrnb(s.ts, control=cntrl)
plot(glr.ts, xaxis.labelFormat=NULL, dx.upperbound=0.5)

#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 <- glrnb(s.ts, control=cntrl2)
plot(glr.ts2, xaxis.labelFormat=NULL)
```

`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 sequentially 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 fitted HMMs are 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 `noStates`-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, ..., $`noStates`.

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

`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 list `control$hmms`, which contains the `"msm"` objects with the fitted HMMs (if `saveHMMs=TRUE`).

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

```
#Simulate outbreak data from HMM
set.seed(123)
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)

if (surveillance.options("allExamples") && requireNamespace("msm")) {
#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.opts=list(x="topright"))
}

if (requireNamespace("msm")) withAutoprint({

#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"
mps <- msm::viterbi.msm(surv$control$hmms[[1]])$fitted
mps

#How often correct?
tab <- data.frame(truth=counts$state + 1, hmm=mps)
table(tab)

})
```

`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$ret == "cases"` then a brute force numerical search for the number needed before alarm (NNBA) is performed. That is, given the past observations, what's the minimum number which would have caused an alarm? Note: Computing this might take a while because the search is done by sequentially increasing/decreasing the last observation by one for each time point in `control$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

`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 M. Frisen and L. Schiöler

Source

The code is an extended R port of the Java code by Marianne Frisén and Linus Schiöler from the Computer Assisted Search For Epidemics (CASE) project, formerly available from `https://case.folkhalsomyndigheten` under the GNU GPL License v3.

An additional feature of the R code is that it contains a search for NNBA (see details).

References

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

Frisén, M. and Andersson, E., (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 <- sts(y, alarm=y*0)

#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(sts, penalty = 20)
```

Arguments

`sts`	object of class `survRes` or `sts`, 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(`*next outbreak beginning*`, 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

an object of class "algoQV", which is a 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 list of quality values
quality <- algo.quality(survResObj)
quality # the list is printed in matrix form


# Format as an "xtable", which is printed with LaTeX markup (by default)
if (requireNamespace("xtable"))
  xtable::xtable(quality)
```

---

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 should 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

`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 $h/2$ 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 $c_t = h/h_t$; 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 $S_0 = h/2$ 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 $h/2$ (i.e. `limit=`$h/2$). 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 (seasonal Poisson)
set.seed(123)
t <- 1:300
lambda <- exp(-0.5 + 0.4 * sin(2*pi*t/52) + 0.6 * cos(2*pi*t/52))
data <- sts(observed = rpois(length(lambda), lambda))

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

# convert to legacy "disProg" class and apply modified Poisson CUSUM
disProgObj <- sts2disProg(data)
res <- algo.rogerson(disProgObj, control=c(hVals, list(theta0t=lambda, range=1:300)))
plot(res, xaxis.years = FALSE)
```

---

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

a 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) )
```

`all.equal` *Test if Two Model Fits are (Nearly) Equal*

Description

Two model fits are compared using standard `all.equal`-methods after discarding certain elements considered irrelevant for the equality of the fits, e.g., the runtime and the call.

Usage

```
## S3 method for class 'twinstim'
all.equal(target, current, ..., ignore = NULL)

## S3 method for class 'hhh4'
all.equal(target, current, ..., ignore = NULL)
```

Arguments

`target, current`	the model fits to be compared.
`...`	further arguments for standard `all.equal`-methods, e.g., the numerical `tolerance`.
`ignore`	an optional character vector of elements to ignore when comparing the two fitted objects. The following elements are always ignored: `"runtime"` and `"call"`.

Value

Either TRUE or a character vector describing differences between the target and the current model fit.

Author(s)

Sebastian Meyer

---

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, animate.epidataCS, and animate.sts for the animation of surveillance data.

---

anscombe.residuals — *Compute Anscombe Residuals*

Description

Compute Anscombe residuals from a fitted glm, which makes them approximately standard normal distributed.

Usage

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

Arguments

m	a fitted "glm"
phi	the current estimated overdispersion

Value

The 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 bootstrap 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. simultaneous 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 and Sebastian Meyer for `eq3a.method = "C"`

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
Ysts <- sts(Y)

#Plot the outbreak
plot(Ysts, 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.opts=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")!
sts.bp <- backprojNP(Ysts, 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 bootstrap CIs
bpnp.control2 <- modifyList(bpnp.control, list(hookFun=NULL, k=2,
  B=10, # in practice, use B >= 1000 !
```

```
  eq3a.method="C"))
sts.bp2 <- backprojNP(Ysts, 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`, `bestCombination` 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

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

---

boda *Bayesian Outbreak Detection Algorithm (BODA)*

Description

The function takes `range` values of a univariate 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. The Bayesian Outbreak Detection Algorithm (`boda`) is due to Manitz and Höhle (2013) and its implementation is illustrated in Salmon et al. (2016). However, `boda` should be considered as an experiment, see the Warning section below!

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,
    samplingMethod=c('joint','marginals'),
    quantileMethod=c("MC","MM")
))
```

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` Data frame (or matrix) of covariates with as many rows as there are time points, i.e., `nrow(sts)`.
- `trend` Boolean indicating whether a linear trend term should be included in the model for the expectation the log-scale
- `season` Boolean to indicate whether a cyclic spline should be included.
- `prior` Model for temporal smoothing using an `f(time, model)` term in the `inla` formula.
- `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.

`samplingMethod` Should one sample from the parameters joint distribution (joint) or from their respective marginal posterior distribution (marginals)?

quantileMethod Character, either `MC` or `MM`. Indicates how to compute the quantile based on the posterior distribution (no matter the inference method): either by sampling `mc.munu` values from the posterior distribution of the parameters and then for each sampled parameters vector sampling `mc.y` response values so that one gets a vector of response values based on which one computes an empirical quantile (MC method, as explained in Manitz and Höhle 2013); or by sampling `mc.munu` from the posterior distribution of the parameters and then compute the quantile of the mixture distribution using bisectioning, which is faster.

Value

the `sts` object with updated `alarms` and `upperbound`, limited to `control$range`.

Warning

This function is currently experimental!! It also heavily depends on the **INLA** package so changes there might affect the operational ability of this function. Since the computations for the Bayesian GAM are quite involved do not expect this function to be particularly fast.

Results are not reproducible if **INLA** uses parallelization (as by default); set `INLA::inla.setOption(num.threads = "1:1")` to avoid that, then do `set.seed` as usual.

Note

This function requires the R package **INLA**, which is currently *not* available from CRAN. It can be obtained from INLA's own repository via `install.packages("INLA", repos="https://inla.r-inla-download.org/R/`

Author(s)

J. Manitz, M. Höhle, M. Salmon

References

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

Salmon, M., Schumacher, D. and Höhle, M. (2016): Monitoring count time series in R: Aberration detection in public health surveillance. *Journal of Statistical Software*, **70** (10), 1-35. doi:10.18637/jss.v070.i10

Examples

```
## Not run:
  ## running this example takes a couple of minutes

  #Load the campylobacteriosis data for Germany
```

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

  #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,
                  samplingMethod = "joint")

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

  plot(cam.boda1, xlab='time [weeks]', ylab='No. reported', dx.upperbound=0)

## End(Not run)
```

---

bodaDelay *Bayesian Outbreak Detection in the Presence of Reporting Delays*

Description

The function takes `range` values of the surveillance time series `sts` and for each time point uses a Bayesian model of the negative binomial family with log link inspired by the work of Noufaily et al. (2012) and of Manitz and Höhle (2014). It allows delay-corrected aberration detection as explained in Salmon et al. (2015). A `reportingTriangle` has to be provided in the `control` slot.

Usage

```
bodaDelay(sts, control = list(
  range = NULL, b = 5, w = 3, mc.munu = 100, mc.y = 10,
  pastAberrations = TRUE, verbose = FALSE,
  alpha = 0.05, trend = TRUE, limit54 = c(5,4),
  inferenceMethod = c("asym","INLA"), quantileMethod = c("MC","MM"),
  noPeriods = 1, pastWeeksNotIncluded = NULL, delay = FALSE))
```

Arguments

`sts`	sts-object to be analysed. Needs to have a reporting triangle.
`control`	list of control arguments: `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.

`range` Specifies the index of all timepoints which should be tested. If range is `NULL` all possible timepoints are used.

`pastAberrations` Boolean indicating whether to include an effect for past outbreaks in a second fit of the model. This option only makes sense if `inferenceMethod` is `INLA`, as it is not supported by the other inference method.

`verbose` Boolean specifying whether to show extra debugging information.

`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

`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).

`inferenceMethod` Which inference method used, as defined in Salmon et al. (2015). If one chooses `"INLA"` then inference is performed with INLA. If one chooses `"asym"` (default) then the asymptotic normal approximation of the posteriori is used.

`pastWeeksNotIncluded` Number of past weeks to ignore in the calculation. The default (`NULL`) means to use the value of `control$w`.

`delay` Boolean indicating whether to take reporting delays into account.

`mc.munu` Number of samples for the parameters of the negative binomial distribution for calculating a threshold

`mc.y` Number of samples for observations when performing Monte Carlo to calculate a threshold

`limit54` c(cases,period) is a vector allowing the user to change these numbers.

`quantileMethod` Character, either `"MC"` (default) or `"MM"`. Indicates how to compute the quantile based on the posterior distribution (no matter the inference method): either by sampling `mc.munu` values from the posterior distribution of the parameters and then for each sampled parameters vector sampling `mc.y` response values so that one gets a vector of response values based on which one computes an empirical quantile (MC method, as explained in Salmon et al. 2015); or by sampling `mc.munu` from the posterior distribution of the parameters and then compute the quantile of the mixture distribution using bisectioning, which is faster.

Value

the `sts` object with updated `alarms` and `upperbound`, limited to `control$range`.

References

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

Noufaily, A., Enki, D.G., Farrington, C.P., Garthwaite, P., Andrews, N.J., Charlett, A. (2012): An improved algorithm for outbreak detection in multiple surveillance systems. Statistics in Medicine, 32 (7), 1206-1222.

Salmon, M., Schumacher, D., Stark, K., Höhle, M. (2015): Bayesian outbreak detection in the presence of reporting delays. Biometrical Journal, 57 (6), 1051-1067.

Examples

```
## Not run:
data("stsNewport")
salm.Normal <- list()
salmDelayAsym <- list()
for (week in 43:45){
  listWeeks <- as.Date(row.names(stsNewport@control$reportingTriangle$n))
  dateObs <- listWeeks[isoWeekYear(listWeeks)$ISOYear==2011 &
                      isoWeekYear(listWeeks)$ISOWeek==week]
  stsC <- sts_observation(stsNewport,
                          dateObservation=dateObs,
                          cut=TRUE)
  inWeeks <- with(isoWeekYear(epoch(stsC)),
                 ISOYear == 2011 & ISOWeek >= 40 & ISOWeek <= 48)

  rangeTest <- which(inWeeks)
  alpha <- 0.07

  # Control slot for Noufaily method
  controlNoufaily <- list(range=rangeTest,noPeriods=10,
                          b=4,w=3,weightsThreshold=2.58,pastWeeksNotIncluded=26,
                          pThresholdTrend=1,thresholdMethod="nbPlugin",alpha=alpha*2,
                          limit54=c(0,50))

  # Control slot for the Proposed algorithm with D=0 correction
  controlNormal <- list(range = rangeTest, b = 4, w = 3,
                        reweight = TRUE, mc.munu=10000, mc.y=100,
                        verbose = FALSE,
                        alpha = alpha, trend = TRUE,
                        limit54=c(0,50),
                        noPeriods = 10, pastWeeksNotIncluded = 26,
                        delay=FALSE)

  # Control slot for the Proposed algorithm with D=10 correction
  controlDelayNorm <-  list(range = rangeTest, b = 4, w = 3,
                            reweight = FALSE, mc.munu=10000, mc.y=100,
                            verbose = FALSE,
                            alpha = alpha, trend = TRUE,
                            limit54=c(0,50),
                            noPeriods = 10, pastWeeksNotIncluded = 26,
                            delay=TRUE,inferenceMethod="asym")

  set.seed(1)
  salm.Normal[[week]] <- farringtonFlexible(stsC, controlNoufaily)
  salmDelayAsym[[week]] <- bodaDelay(stsC, controlDelayNorm)
}

opar <- par(mfrow=c(2,3))
lapply(salmDelayAsym[c(43,44,45)],plot, legend=NULL, main="", ylim=c(0,35))
```

```
lapply(salm.Normal[c(43,44,45)],plot, legend=NULL, main="", ylim=c(0,35))
par(opar)

## End(Not run)
```

---

calibrationTest *Calibration Tests for Poisson or Negative Binomial Predictions*

Description

The implemented calibration tests for Poisson or negative binomial predictions of count data are based on proper scoring rules and described in detail in Wei and Held (2014). The following proper scoring rules are available: Dawid-Sebastiani score (`"dss"`), logarithmic score (`"logs"`), ranked probability score (`"rps"`).

Usage

```
calibrationTest(x, ...)

## Default S3 method:
calibrationTest(x, mu, size = NULL,
                which = c("dss", "logs", "rps"),
                tolerance = 1e-4, method = 2, ...)
```

Arguments

`x`	the observed counts. All involved functions are vectorized and also accept matrices or arrays.
`mu`	the means of the predictive distributions for the observations `x`.
`size`	either `NULL` (default), indicating Poisson predictions with mean `mu`, or dispersion parameters of negative binomial forecasts for the observations `x`, parametrized as in `dnbinom` with variance `mu*(1+mu/size)`.
`which`	a character string indicating which proper scoring rule to apply.
`tolerance`	absolute tolerance for the null expectation and variance of `"logs"` and `"rps"`. For the latter, see the note below. Unused for `which = "dss"` (closed form).
`method`	selection of the z-statistic: `method = 2` refers to the alternative test statistic Z_s^* of Wei and Held (2014, Discussion), which has been recommended for low counts. `method = 1` corresponds to Equation 5 in Wei and Held (2014).
`...`	unused (argument of the generic).

Value

an object of class `"htest"`, which is a list with the following components:

`method`	a character string indicating the type of test performed (including `which` scoring rule).

`data.name`	a character string naming the supplied `x` argument.
`statistic`	the z-statistic of the test.
`parameter`	the number of predictions underlying the test, i.e., `length(x)`.
`p.value`	the p-value for the test.

Note

If the **gsl** package is installed, its implementations of the Bessel and hypergeometric functions are used when calculating the null expectation and variance of the `rps`. These functions are faster and yield more accurate results (especially for larger `mu`).

Author(s)

Sebastian Meyer and Wei Wei

References

Wei, W. and Held, L. (2014): Calibration tests for count data. *Test*, **23**, 787-805.

Examples

```
mu <- c(0.1, 1, 3, 6, pi, 100)
size <- 0.1
set.seed(1)
y <- rnbinom(length(mu), mu = mu, size = size)
calibrationTest(y, mu = mu, size = size) # p = 0.99
calibrationTest(y, mu = mu, size = 1) # p = 4.3e-05
calibrationTest(y, mu = 1, size = size) # p = 0.6959
calibrationTest(y, mu = 1, size = size, which = "rps") # p = 0.1286
```

`campyDE` *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 have been queried from the Survstat@RKI database of the German Robert Koch Institute (https://survstat.rki.de/).

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

A complete data description and an analysis of the data can be found in Manitz and Höhle (2013).

References

Manitz, J. and Höhle, M. (2013): Bayesian outbreak detection algorithm for monitoring reported cases of campylobacteriosis in Germany. Biometrical Journal, 55(4), 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 <- sts(epoch=campyDE$date, observed=campyDE$case, state=campyDE$state)

# 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 as described by Höhle (2010). Logistic, multinomial logistic, proportional odds or Bradley-Terry regression models are used to specify in-control and out-of-control parameters. The implementation is illustrated in Salmon et al. (2016).

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 (PMF) 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, e.g, 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 the values in `c("cases","value")`. Note: For the binomial PMF it is possible to compute this value explicitly, which is much faster than the numeric search otherwise conducted. In case `dfun` just corresponds to `dbinom` just set the attribute `isBinomialPMF` for the `dfun` object.

`...` 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 reset (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): Online Change-Point Detection in Categorical Time Series. In: T. Kneib and G. Tutz (Eds.), Statistical Modelling and Regression Structures, Physica-Verlag.

Salmon, M., Schumacher, D. and Höhle, M. (2016): Monitoring count time series in R: Aberration detection in public health surveillance. *Journal of Statistical Software*, **70** (10), 1-35. doi:10.18637/jss.v070.i10

See Also

`LRCUSUM.runlength`

Examples

```
have_GAMLSS <- require("gamlss")

if (have_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)

  #Replace the observed and epoch column names to something more convenient
  dict <- c("observed"="y", "epoch"="t", "population"="n")
```

```
replace <- dict[colnames(abattoir.df)]
colnames(abattoir.df)[!is.na(replace)] <- replace[!is.na(replace)]

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 <- 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],dx.upperbound=0)
  lines(pi0,col="green")
  lines(pi1,col="red")

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

---

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

```
data("hagelloch")
fit <- twinSIR(~ household, data = hagelloch)  # a simplistic model
## extract the "residual process", i.e., the fitted cumulative intensities
residuals(fit)
## assess goodness of fit based on these residuals
checkResidualProcess(fit)  # could be better
```

---

clapply *Conditional* lapply

Description

Use lapply if the input is a list and otherwise apply the function directly to the input *and* wrap the result in a list. The function is implemented as

```
if (is.list(X)) lapply(X, FUN, ...) else list(FUN(X, ...))
```

Usage

```
clapply(X, FUN, ...)
```

Arguments

X	a list or a single R object on which to apply FUN.
FUN	the function to be applied to (each element of) X.
...	optional arguments to FUN.

Value

a list (of length 1 if X is not a list).

`coeflist` *List Coefficients by Model Component*

Description

S3-generic function to use with models which contain several groups of coefficients in their coefficient vector. The `coeflist` methods are intended to list the coefficients by group. The default method simply `split`s the coefficient vector given the number of coefficients by group.

Usage

```
coeflist(x, ...)

## Default S3 method:
coeflist(x, npars, ...)
```

Arguments

`x`	a model with groups of coefficients or, for the default method, a vector of coefficients.
`npars`	a named vector specifying the number of coefficients per group.
`...`	potential further arguments (currently ignored).

Value

a list of coefficients

Author(s)

Sebastian Meyer

Examples

```
## the default method just 'split's the coefficient vector
coefs <- c(a = 1, b = 3, dispersion = 0.5)
npars <- c(regression = 2, variance = 1)
coeflist(coefs, npars)
```

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(deleval)
```

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" from package **sp**, "owin" from package **spatstat.geom**, or "gpc.poly" from **gpclib** (if available).

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.geom**'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" and this S4 class is not yet registered in the current R session (by loading **gpclib** beforehand), only the pts slot of a "gpc.poly" is returned with a warning.

See Also

disc in package **spatstat.geom**.

Examples

```
## Construct circles with increasing accuracy and of different spatial classes
disc1 <- discpoly(c(0,0), 5, npoly=8, class = "owin")
disc2 <- discpoly(c(0,0), 5, npoly=16, class = "Polygon")
disc3 <- discpoly(c(0,0), 5, npoly=32, class = "gpc.poly")  # may warn

## Look at the results
plot(disc1, axes=TRUE, main="", border=2, lwd=2)
lines(disc2, col=3, lwd=2)

str(disc3)  # a list or a formal "gpc.poly" (if gpclib is available)
if (is(disc3, "gpc.poly")) {
```

```
  plot(disc3, add=TRUE, poly.args=list(border=4))
} else {
  lines(disc3[[1]], col=4, lwd=2)
}

## to only _draw_ a circle
symbols(0, 0, circles=5, inches=FALSE, add=TRUE, lwd=2, lty=3)
```

`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`	an object of class `"disProg"`
`map`	an optional `"SpatialPolygons"` object
`sts`	an 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 and its extensions*

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 historic values, since it only needs counts from the recent past.

Usage

```
earsC(sts, control = list(range = NULL, method = "C1",
                          baseline = 7, minSigma = 0,
                          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 in the `sts` object of all the timepoints which should be monitored. If `range` is `NULL` the maximum number of possible timepoints is used (this number depends on the method chosen):
  - **C1** all timepoints from the observation with index `baseline + 1` can be monitored,
  - **C2** timepoints from index `baseline + 3` can be monitored,
  - **C3** timepoints starting from the index `baseline + 5` can be monitored.
- `method` String indicating which method to use:
  - `"C1"` for EARS C1-MILD method (Default),
  - `"C2"` for EARS C2-MEDIUM method,
  - `"C3"` for EARS C3-HIGH method.

  See Details for further information about the methods.
- `baseline` how many time points to use for calculating the baseline, see details
- `minSigma` By default 0. If `minSigma` is higher than 0, for C1 and C2, the quantity zAlpha * minSigma is then the alerting threshold if the baseline is zero. Howard Burkom suggests using a value of 0.5 or 1 for sparse data.
- `alpha` An approximate (two-sided) $(1-\alpha)\cdot 100\%$ prediction interval is calculated. By default if `alpha` is `NULL` the value 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 the observed counts in 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 the C2 method. This method has similarities with a CUSUM method due to it 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 is presented as such.

Here is what the function does for each method, see the literature sources for further details:

1. For C1 the baseline are the `baseline` (default 7) timepoints before the assessed timepoint t, t-`baseline` 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
$$\bar{Y}_1(t) = \frac{1}{\texttt{baseline}} \sum_{i=t-1}^{t-\texttt{baseline}} Y(i)$$
and
$$S_1^2(t) = \frac{1}{6} \sum_{i=t-1}^{t-\texttt{baseline}} [Y(i) - \bar{Y}_1(i)]^2.$$
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 standard normal distribution.

   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 similar to C1 apart from a 2-day lag in the baseline definition. In other words the baseline for C2 is `baseline` (Default: 7) timepoints with a 2-day lag before the monitored timepoint t, i.e. $(t - \texttt{baseline} - 2)$ 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

$$\bar{Y}_2(t) = \frac{1}{\texttt{baseline}} \sum_{i=t-3}^{t-\texttt{baseline}-2} Y(i)$$

and

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

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 standard normal distribution.

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 consists of the timepoints $t - (\texttt{baseline} + 4)$ 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 standard normal distribution.

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, H. Burkom

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.

Salmon, M., Schumacher, D. and Höhle, M. (2016): Monitoring count time series in R: Aberration detection in public health surveillance. *Journal of Statistical Software*, **70** (10), 1-35. doi:10.18637/jss.v070.i10

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


# Compare C3 upperbounds depending on alpha
res3 <- earsC(stsObj, control = list(range = 20:208,method="C3",alpha = 0.001))
plot(upperbound(res3), type='l')
res3 <- earsC(stsObj, control = list(range = 20:208,method="C3"))
lines(upperbound(res3), col='red')
```

---

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`. Their implementation is illustrated in Meyer et al. (2017, Section 4), see `vignette("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.

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(), D = dist,
           max.time = NULL, 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(), D = dist, .latent = FALSE, ...)

## 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(), D = dist, ...)
```

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")` in package **survival**, with additional information (variables) along the process. Rows will be sorted automatically during conversion. The observation period is split 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, max.time` observation period. In the resulting `"epidata"`, the time scale will be relative to the start time `t0`. Individuals that have already been removed prior to `t0`, i.e., rows with `tR <= t0`, will be dropped. The end of the observation period

(max.time) will by default (NULL, or if NA) coincide with the last observed event.

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).
CAVE: Support for latent periods (tE.col) is experimental! twinSIR cannot handle them anyway.

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 numeric (by column number), character (by column name), or NULL (no coordinates, e.g., if D is a pre-specified distance matrix). These columns contain the individuals' coordinates, which determine the distance matrix for the distance-based components of the force of infection (see argument f). By default, Euclidean distance is used (see argument D).
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 animate-method requires coordinates.

`f` a *named* list of *vectorized* functions for a distance-based force of infection. The functions must interact elementwise on a (distance) matrix `D` so that `f[[m]](D)` results in a matrix. A simple example is `function(u) {u <= 1}`, which indicates if the Euclidean distance between the individuals is smaller than or equal to 1. The names of the functions determine 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: Let $I(t)$ denote the set of infectious individuals just before time t. Then, for individual i at time t, the m'th covariate has the value $\sum_{j \in I(t)} f_m(d_{ij})$, where d_{ij} denotes entries of the distance matrix (by default this is the Euclidean distance $||s_i - s_j||$ between the individuals' coordinates, but see argument `D`).

`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]]`.

`D` either a function to calculate the distances between the individuals with locations taken from `coord.cols` (the default is Euclidean distance via the function `dist`) and the result converted to a matrix via `as.matrix`, or a pre-computed distance matrix with `dimnames` containing the individual ids (a classed `"Matrix"` is supported).

`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.

`.latent` (internal) logical indicating whether to allow for latent periods (EXPERIMENTAL). Otherwise (default), the function verifies that an event (i.e., switching to the I state) only happens when the respective individual is at risk (i.e., in the S state).

`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

References

Meyer, S., Held, L. and Höhle, M. (2017): Spatio-temporal analysis of epidemic phenomena using the R package **surveillance**. *Journal of Statistical Software*, **77** (11), 1-55. doi:10.18637/jss.v077.i11

See Also

The hagelloch data as an example.

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

```
data("hagelloch")   # see help("hagelloch") for a description
head(hagelloch.df)

## convert the original data frame to an "epidata" event history
myEpi <- as.epidata(hagelloch.df, t0 = 0,
                    tI.col = "tI", tR.col = "tR", id.col = "PN",
                    coords.cols = c("x.loc", "y.loc"),
                    keep.cols = c("SEX", "AGE", "CL"))


str(myEpi)
head(as.data.frame(myEpi))  # "epidata" has event history format
summary(myEpi)              # see 'summary.epidata'
plot(myEpi)                 # see 'plot.epidata' and also 'animate.epidata'


## add distance- and covariate-based weights for the force of infection
## in a twinSIR model, see vignette("twinSIR") for a description
myEpi <- update(myEpi,
    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
    )
)
## this is now identical to the prepared hagelloch "epidata"
stopifnot(all.equal(myEpi, hagelloch))
```

`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 (overlaid) in the same plotting region as time proceeds. 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 print to files, `time.spacing` must not be `NULL`, a screen device must be available, 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,tp),"
sep="")))
```

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("hagelloch")
(s <- summary(hagelloch))

# plot the ordering of the events only
animate(s)   # or: animate(hagelloch)

# with timer (animate only up to t=10)
animate(s, time.spacing=0.1, end=10, sleep=0.01,
        legend.opts=list(x="topleft"))

# Such an animation can be saved in various ways using tools of
# the animation package, e.g., saveHTML()
if (interactive() && 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, legend.opts=list(x="topleft"),
            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 split 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("hagelloch")
subset(hagelloch, start < 25 & stop > 25 & id %in% 9:13, select = 1:7)
# there is no "stop" time at 25, but we can add this extra stop
nrow(hagelloch)
moreStopsEpi <- intersperse(hagelloch, stoptimes = 25)
nrow(moreStopsEpi)
subset(moreStopsEpi, (stop == 25 | start == 25) & id %in% 9:13, select = 1:7)
```

---

epidata_plot *Plotting the Evolution of an Epidemic*

Description

Functions for plotting the evolution of epidemics. The `plot` methods for `class`es `"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*usceptible, 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("hagelloch")
(s <- summary(hagelloch))

# rudimentary stateplot
stateplot(s, id = "187")

# evolution of the epidemic
plot(s)
```

`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 ids 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("hagelloch")
s <- summary(hagelloch)
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)
```

---

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 endemic-epidemic intensity models with the function twinstim (Meyer et al., 2012). The implementation is described in Meyer et al. (2017, Section 3), see vignette("twinstim").

Usage

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

getSourceDists(object, dimension = c("space", "time"))
```

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 there cannot be events beyond `stgrid` (i.e., `time<=T` is required). Events at or before time t_0 = `min(stgrid$start)` are allowed and form the prehistory of the process.

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).

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`. Prior simplification of `W` may considerably reduce the computational burden of likelihood evaluations in `twinstim` models with non-trivial spatial interaction functions (see 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, see `discpoly`.

`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.geom**). In **surveillance** <= 1.6-0, package **gpclib** was used; this is no longer supported, neither is **rgeos**.

`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"`.
dimension	the distances of all events to their potential source events can be computed in either the `"space"` or `"time"` dimension.

Details

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

The `[`-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. The `[`-method is used in `subset.epidataCS`, which is implemented similar to `subset.data.frame`.

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 (possibly truncated at `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.geom** 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.geom** 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

Since the observation region W defines the integration domain in the point process likelihood, the more detailed the polygons of W are the longer it will take to fit a twinstim. You are advised to sacrifice some shape details for speed by reducing the polygon complexity, for example via the mapshaper JavaScript library wrapped by the R package **rmapshaper**, or via simplify.owin.

Author(s)

Sebastian Meyer

Contributions to this documentation by Michael Höhle and Mayeul Kauffmann.

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:10.1111/j.15410420.2011.01684.x

Meyer, S., Held, L. and Höhle, M. (2017): Spatio-temporal analysis of epidemic phenomena using the R package **surveillance**. *Journal of Statistical Software*, **77** (11), 1-55. doi:10.18637/jss.v077.i11

See Also

vignette("twinstim")

plot.epidataCS for plotting, and animate.epidataCS for the animation of such an epidemic. There is also an update method for the "epidataCS" class.

To re-extract the events point pattern from "epidataCS", use as(object, "SpatialPointsDataFrame").

It is possible to convert an "epidataCS" point pattern to an "epidata" object (as.epidata.epidataCS), or to aggregate the events into an "sts" object (epidataCS2sts).

Examples

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

## print and summary
print(imdepi, n=5, digits=2)

print(s <- summary(imdepi))

plot(s$counter,  # same as 'as.stepfun(imdepi)'
     xlab = "Time [days]", ylab="Number of infectious individuals",
     main=paste("Time course of the number of infectious individuals",
                "assuming an infectious period of 30 days", sep="\n"))
plot(table(s$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)

## a histogram of the spatial distances to potential source events
## (i.e., to events of the previous eps.t=30 days within eps.s=200 km)
sourceDists_space <- getSourceDists(imdepi, "space")
hist(sourceDists_space); rug(sourceDists_space)

## internal structure of an "epidataCS"-object
str(imdepi, max.level=4)
## see help("imdepi") for more info on the data set

## extraction methods subset the 'events' component
imdepi[101:200,]
head(imdepi, n=1)           # only first event
tail(imdepi, n=4)           # only last 4 events
subset(imdepi, type=="B")   # only events of type B

## see help("plot.epidataCS") for convenient plot-methods for "epidataCS"


###
### reconstruct the "imdepi" object
###

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

## extract point pattern of events from the "imdepi" data
## a) as a data frame with coordinate columns via marks()
eventsData <- marks(imdepi)
head(eventsData)
## b) as a Spatial object via the coerce-method
```

```
events <- as(imdepi, "SpatialPointsDataFrame")
head(events)

## 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)

## space-time grid with endemic covariates
head(stgrid <- imdepi$stgrid[,-1])

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

---

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. The start specification should reflect the beginning of object$stgrid, i.e., the start of the first time interval.
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

Conversion to "sts" only makes sense if the time intervals (BLOCKs) of the stgrid are regularly spaced (to give freq intervals per year). Note that events of the prehistory (not covered by stgrid) are not included in the resulting sts object.

Some comments on the conversion 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
imdsts <- epidataCS2sts(imdepi, freq = 12, start = c(2002, 1),
                        neighbourhood = NULL, # not needed here
                        tiles = districtsD)

## check the overall number of events by district
stopifnot(all.equal(colSums(observed(imdsts)),
                    c(table(imdepi$events$tile))))

## compare plots of monthly number of cases
opar <- par(mfrow = c(2, 1))
plot(imdepi, "time")
plot(imdsts, ~time)
par(opar)

## plot number of cases by district in Bavaria (municipality keys 09xxx)
imd09 <- imdsts[, grep("^09", colnames(imdsts), value = TRUE), drop = TRUE]
plot(imd09, ~unit)

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

`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 occurred. 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 specifying 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 occurring (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 for "SpatialPolygons".

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")

## Not run:
# 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)

## End(Not run)

# Such an animation can be saved in various ways using the tools of
# the animation package, e.g., saveHTML()
if (interactive() && 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_permute *Randomly Permute Time Points or Locations of* "epidataCS"

Description

Monte Carlo tests for space-time interaction (epitest) use the distribution of some test statistic under the null hypothesis of no space-time interaction. For this purpose, the function permute.epidataCS randomly permutes the time or space labels of the events.

Usage

```
permute.epidataCS(x, what = c("time", "space"), keep)
```

Arguments

x	an object of class "epidataCS".
what	character string determining what to permute: time points (default) or locations.
keep	optional logical expression to be evaluated in the context of x$events@data, determining for which events the time and location should be kept as is. For instance, to keep some "prehistory" before time point 30 unchanged, use keep = time <= 30.

Value

the permuted "epidataCS" object.

Author(s)

Sebastian Meyer

See Also

epitest

Examples

```
data("imdepi")

set.seed(3)
permepi <- permute.epidataCS(imdepi, what = "time", keep = time <= 30)

print(imdepi, n = 8)
print(permepi, n = 8)
## the first 6 events are kept (as are all row.names),
## the time labels of the remaining events are shuffled
## (and events then again sorted by time),
## the marginal temporal distribution is unchanged
```

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). The spatial plot can be enriched with tile-specific color levels to indicate attributes such as the population (using spplot).

Usage

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

epidataCSplot_time(x, subset, by = type,
                   t0.Date = NULL, breaks = "stgrid", 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, by = type, tiles = x$W, pop = NULL,
                    cex.fun = sqrt, points.args = list(), add = FALSE,
                    legend.types = list(), legend.counts = list(),
                    sp.layout = NULL, ...)
```

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 (or the tiles) 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 tiles.

by	an expression evaluated in `marks(x)`, defining how events should be stratified in the plot (the result is converted to a factor), or `NULL` to disregard event types. By default (`by = type`) the plot distinguishes between event types, i.e., the bars of the temporal plot are stacked by type, and the point colors in the spatial plot differ by type, respectively. Note: to select specific event types for plotting use the `subset` argument, e.g., `subset=(type=="B")`.
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, 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).
breaks	a specification of the histogram break points, see `hist` (or `hist.Date` if `t0.Date` is used). The default value `"stgrid"` is special and means to use the temporal grid points `with(x$stgrid, c(start[1L], unique.default(stop)))` as breaks (or their `"Date"` equivalents).
freq	see `hist`, defaults to `TRUE`.
col	fill colour for the bars of the histogram, defaults to the vector of `rainbow` colours.
cumulative	if a list (of style options), lines for the cumulative number of events (per type) will be added to the plot. Possible options are `axis` (logical), `lab` (axis label), `maxat` (single integer affecting the axis range), `lwd`, `col`, and `offset` (a numeric vector of length the number of types).
add	logical (default: `FALSE`) indicating if the plot should be added to an existing window. Ignored if an `spplot` is created (if `pop` is non-`NULL`).
mar	see `par`. The default (`NULL`) is `mar <- par("mar")`, with `mar[4] <- mar[2]` if an axis is requested for the `cumulative` numbers.
xlim, ylim	`NULL` provides automatic axis limits.
xlab, ylab	axis labels (with sensible defaults).
main	main title of the plot (defaults to no title).
panel.first	expression that should be evaluated after the plotting window has been set up but before the histogram is plotted. Defaults to adding horizontal grid lines.
legend.types	if a list (of arguments for `legend`), a legend for the event types is added to the plot in case there is more than one type.
tiles	the observation region `x$W` (default) or, alternatively, a `"SpatialPolygons"` representation of the tiles of `x$stgrid`.
pop	if `tiles` is a `"SpatialPolygonsDataFrame"`, `pop` can specify an attribute to be displayed in a `levelplot` behind the point pattern, see `spplot`. By default (`NULL`), the conventional graphics system is used to display the `tiles` and event locations, otherwise the result is a `trellis.object`.
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.

sp.layout — optional list of additional layout items in case pop is non-NULL, see spplot.

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, or the trellis.object generated by spplot (if pop is non-NULL).

Author(s)

Sebastian Meyer

See Also

animate.epidataCS

Examples

```
data("imdepi")

## show the occurrence of events along time
plot(imdepi, "time", main = "Histogram of event time points")
plot(imdepi, "time", by = NULL, main = "Aggregated over both event types")
plot(imdepi, "time", subset = type == "B", main = "Only type B")

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

## with the district-specific population density in the background,
## a scale bar, and customized point style
load(system.file("shapes", "districtsD.RData", package = "surveillance"))
districtsD$log10popdens <- log10(districtsD$POPULATION/districtsD$AREA)
keylabels <- (c(1,2,5) * rep(10^(1:3), each=3))[-1]
plot(imdepi, "space", tiles = districtsD, pop = "log10popdens",
     ## modify point style for better visibility on gray background
     points.args = list(pch=c(1,3), col=c("orangered","blue"), lwd=2),
     ## metric scale bar, see proj4string(imdepi$W)
     sp.layout = layout.scalebar(imdepi$W, scale=100, labels=c("0","100 km")),
     ## gray scale for the population density and white borders
     col.regions = gray.colors(100, start=0.9, end=0.1), col = "white",
     ## color key is equidistant on log10(popdens) scale
     at = seq(1.3, 3.7, by=0.05),
     colorkey = list(labels=list(at=log10(keylabels), labels=keylabels),
                     title=expression("Population density per " * km^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 determining possible transmission between the event types, the numerical accuracy nCircle2Poly of the polygonal approximation, and the endemic covariates from stgrid (including the time intervals). 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, the influence regions, or the endemic covariates copied from the new stgrid.

Usage

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

Arguments

object	an object of class "epidataCS".
eps.t	numeric vector of length 1 or corresponding to the number of events in object$events. The event data column eps.t specifies the maximum temporal influence radius (e.g., length of infectious period, time to culling, etc.) of the events.
eps.s	numeric vector of length 1 or corresponding to the number of events in object$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.
stgrid	a new data.frame with endemic covariates, possibly transformed from or adding to the original object$stgrid. The grid must cover the same regions as the original, i.e., levels(object$stgrid$tile) must remain identical. See epidataCS for a detailed description of the required format.
...	unused (argument of the generic).

Value

The updated "epidataCS" object.

Author(s)

Sebastian Meyer

See Also

class "epidataCS".

Examples

```
data("imdepi")

## modify interaction ranges: shorter but over the whole observation region
imdepi2 <- update(imdepi, eps.t = 20, eps.s = Inf)

## Note: this reduces the number of infectives per time point
plot(as.stepfun(imdepi))
lines(as.stepfun(imdepi2), col = 2)
## and affects the set of potential sources for each event:
table("imdepi"  = summary(imdepi) $nSources,
      "imdepi2" = summary(imdepi2)$nSources)

## switch to a time-constant grid
imdepi3 <- update(imdepi2, stgrid = subset(imdepi2$stgrid, BLOCK == 1, -stop))
imdepi3 # "1 time block"
```

---

fanplot *Fan Plot of Forecast Distributions*

Description

The fanplot() function in **surveillance** wraps functionality of the dedicated **fanplot** package, employing a different default style and optionally adding point predictions and observed values.

Usage

```
fanplot(quantiles, probs, means = NULL, observed = NULL, start = 1,
        fan.args = list(), means.args = list(), observed.args = list(),
        key.args = NULL, xlim = NULL, ylim = NULL, log = "",
        xlab = "Time", ylab = "No. infected", add = FALSE, ...)
```

Arguments

quantiles	a time x probs matrix of forecast quantiles at each time point.
probs	numeric vector of probabilities with values between 0 and 1.
means	(optional) numeric vector of point forecasts.
observed	(optional) numeric vector of observed values.
start	time index (x-coordinate) of the first prediction.
fan.args	a list of graphical parameters for the fan, e.g., to employ a different colorRampPalette as fan.col, or to enable contour lines via ln.
means.args	a list of graphical parameters for lines to modify the plotting style of the means. The default is a white line within the fan.

observed.args	a list of graphical parameters for lines to modify the plotting style of the observed values.
key.args	if a list, a color key (in fan()'s "boxfan"-style) is added to the fan chart. The list may include positioning parameters start (the x-position) and ylim (the y-range of the color key), space to modify the width of the boxfan, and rlab to modify the labels. An alternative way of labeling the quantiles is via the argument ln in fan.args.
xlim, ylim	axis ranges.
log	a character string specifying which axes are to be logarithmic, e.g., log="y" (see plot.default).
xlab, ylab	axis labels.
add	logical indicating if the fan plot should be added to an existing plot.
...	further arguments are passed to plot.default. For instance, panel.first could be used to initialize the plot with grid(nx=NA, ny=NULL) lines.

Value

NULL (invisibly), with the side effect of drawing a fan chart.

Author(s)

Sebastian Meyer

See Also

the underlying fan function in package **fanplot**. The function is used in plot.oneStepAhead and plot.hhh4sims.

Examples

```
## artificial data example to illustrate the graphical options
if (requireNamespace("fanplot")) {
    means <- c(18, 19, 20, 25, 26, 35, 34, 25, 19)
    y <- rlnorm(length(means), log(means), 0.5)
   quantiles <- sapply(1:99/100, qlnorm, log(means), seq(.5,.8,length.out=length(means)))

    ## default style with point predictions, color key and log-scale
    fanplot(quantiles = quantiles, probs = 1:99/100, means = means,
            observed = y, key.args = list(start = 1, space = .3), log = "y")

    ## with contour lines instead of a key, and different colors
    pal <- colorRampPalette(c("darkgreen", "gray93"))
    fanplot(quantiles = quantiles, probs = 1:99/100, observed = y,
            fan.args = list(fan.col = pal, ln = c(5,10,25,50,75,90,95)/100),
            observed.args = list(type = "b", pch = 19))
}
```

farringtonFlexible *Surveillance for Univariate Count Time Series Using an Improved Farrington Method*

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. The implementation is illustrated in Salmon et al. (2016).

Usage

```
farringtonFlexible(sts, control = list(
    range = NULL, b = 5, w = 3,
    reweight = TRUE, weightsThreshold = 2.58,
    verbose = FALSE, glmWarnings = TRUE,
    alpha = 0.05, trend = TRUE, pThresholdTrend = 0.05,
    limit54 = c(5,4), powertrans = "2/3",
    fitFun = "algo.farrington.fitGLM.flexible",
    populationOffset = FALSE,
    noPeriods = 1, pastWeeksNotIncluded = NULL,
    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 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. The default (`NULL`) means to use the value of `control$w`. Setting `pastWeeksNotIncluded=26` might be preferable (Noufaily et al., 2012).

`thresholdMethod` Method to be used to derive the upperbound. Options are `"delta"` for the method described in Farrington et al. (1996), `"nbPlugin"` 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 performed 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 `control` slot of the input `sts` is amended with the following matrix elements, all with `length(range)` rows:

trend Booleans indicating whether a time trend was fitted for this time point.

trendVector coefficient of the time trend in the GLM for this time point. If no trend was fitted it is equal to NA.

pvalue probability of observing a value at least equal to the observation under the null hypothesis .

expected expectation of the predictive distribution for each timepoint. It is only reported if the conditions for raising an alarm are met (enough cases).

mu0Vector input for 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 overdispersion of the GLM at each timepoint.

Author(s)

M. Salmon, M. Höhle

References

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

Noufaily, A., Enki, D.G., Farrington, C.P., Garthwaite, P., Andrews, N.J., Charlett, A. (2012): An improved algorithm for outbreak detection in multiple surveillance systems. Statistics in Medicine, 32 (7), 1206-1222.

Salmon, M., Schumacher, D. and Höhle, M. (2016): Monitoring count time series in R: Aberration detection in public health surveillance. *Journal of Statistical Software*, **70** (10), 1-35. doi:10.18637/jss.v070.i10

See Also

algo.farrington.fitGLM,algo.farrington.threshold

Examples

```
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
control1 <-  list(range=282:312,
                  noPeriods=1,
                  b=4, w=3, weightsThreshold=1,
                  pastWeeksNotIncluded=3,
                  pThresholdTrend=0.05,
                  alpha=0.1)
control2 <- list(range=282:312,
                 noPeriods=10,
                 b=4, w=3, weightsThreshold=2.58,
                 pastWeeksNotIncluded=26,
                 pThresholdTrend=1,
                 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', legend.opts=NULL)
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("topleft",
       legend=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 $\left|\frac{ARL(h^*)-\gamma}{\gamma}\right| <$ `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: https://survstat.rki.de/; 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:10.1214/14AOAS743

Examples

```
data("fluBYBW")

# Count time series plot
plot(fluBYBW, ~time)

# Map of disease incidence (per 100000 inhabitants) for the year 2001
plot(fluBYBW, ~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)
```

`formatDate` *Convert Dates to Character (Including Quarter Strings)*

Description

An extension of `format.Date` with additional formatting strings for quarters. Used by `linelist2sts`.

Usage

```
formatDate(x, format)
```

Arguments

`x`	a `"Date"` object.
`format`	a character string, see `strftime` for possible specifications. Further to these base formats, `formatDate` implements: `"%Q"` the quarter as a numeric `"%OQ"` the quarter as a roman numeral `"%q"` the day within the quarter

Value

a character vector representing the input date(s) `x` following the `format` specification.

See Also

`strftime`

Examples

```
formatDate(as.Date("2021-10-13"), "%G/%OQ/%q")
```

formatPval — *Pretty p-Value Formatting*

Description

Just YAPF – yet another p-value formatter...

It is a wrapper around `format.pval`, such that by default `eps = 1e-4`, `scientific = FALSE`, `digits = if (p<10*eps) 1 else 2`, and `nsmall = 2`.

Usage

```
formatPval(pv, eps = 1e-4, scientific = FALSE, ...)
```

Arguments

`pv`	a numeric vector (of p-values).
`eps`	a numerical tolerance, see `format.pval`.
`scientific`	see `format`.
`...`	further arguments passed to `format.pval` (but `digits` and `nsmall` are hard-coded internally).

Value

The character vector of formatted p-values.

Examples

```
formatPval(c(0.9, 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, $Y_{[t][\boldsymbol{s}],k}$, by time-space-type cell. The rate of the corresponding Poisson distribution is $e_{[t][\boldsymbol{s}]} \cdot \lambda([t],[\boldsymbol{s}],k)$, where $e_{[t][\boldsymbol{s}]} = |[t]||[\boldsymbol{s}]|$ 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", "imdepifit")

## Fit an endemic-only twinstim() and an equivalent model wrapped in glm()
fit_twinstim <- update(imdepifit, epidemic = ~0, siaf = NULL, subset = NULL,
                       optim.args=list(control=list(trace=0)), verbose=FALSE)
fit_glm <- glm_epidataCS(formula(fit_twinstim)$endemic, data = imdepi)

## Compare the coefficients
cbind(twinstim = coef(fit_twinstim), glm = coef(fit_glm))


### 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"))
imdsts <- 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$stgrid, !duplicated(BLOCK)),
                  stop - start), ncol(imdsts)) *
          sum(districtsD$POPULATION) * population(imdsts)

## fit the model using hhh4()
fit_hhh4 <- hhh4(imdsts, control = list(
    end = list(
        f = addSeason2formula(~I(start/365-3.5), period=365, timevar="start"),
        offset = offset
    ), family = "Poisson", subset = 1:nrow(imdsts),
    data = list(start=with(subset(imdepi$stgrid, !duplicated(BLOCK)), start))))

summary(fit_hhh4)
stopifnot(all.equal(coef(fit_hhh4), coef(fit_glm), check.attributes=FALSE))
```

ha — *Hepatitis A in Berlin*

Description

Number of Hepatitis A cases among adult (age>18) males 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 was generated from ha via the converter function disProg2sts and includes a map of Berlin's districts.

Source

Robert Koch-Institut: SurvStat: https://survstat.rki.de/; Queried on 25 August 2006.

Robert Koch Institut, Epidemiologisches Bulletin 33/2006, p.290.

Examples

```
## deprecated "disProg" object
data("ha")
ha
plot(aggregate(ha))

## new-style "sts" object
data("ha.sts")
ha.sts
plot(ha.sts, ~time)  # = plot(aggregate(ha.sts, by = "unit"))
plot(ha.sts, ~unit, labels = TRUE)
```

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). This dataset is used to illustrate the twinSIR model class in vignette("twinSIR").

Usage

```
data("hagelloch")
```

Format

Loading `data("hagelloch")` gives two objects: `hagelloch` and `hagelloch.df`. The latter is the original `data.frame` of 188 rows with individual information for each infected child. `hagelloch` has been generated from `hagelloch.df` via `as.epidata` (see the Examples below) to obtain an `"epidata"` object for use with `twinSIR`. It contains the entire SIR event history of the outbreak (but not all of the covariates).

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 complications, 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 prodromes (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-Recovered (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:

household: the number of currently infectious children in the same household (including the child itself if it is currently infectious).

nothousehold: the number of currently infectious children outside the household.

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.

Such epidemic covariates can been computed by specifying suitable `f` and `w` arguments in `as.epidata` at conversion (see the code below), or at a later step via the `update`-method for `"epidata"`.

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 https://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

data class: `epidata`

point process model: `twinSIR`

illustration with `hagelloch`: `vignette("twinSIR")`

Examples

```
data("hagelloch")
head(hagelloch.df)   # original data documented in Oesterle (1992)
head(as.data.frame(hagelloch))   # "epidata" event history format

## 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="")


### "epidata" summary and plot methods

(s <- summary(hagelloch))
head(s$byID)
plot(s)

## Not run:
  # Show a dynamic illustration of the spread of the infection
  animate(hagelloch, time.spacing=0.1, sleep=1/100,
          legend.opts=list(x="topleft"))

## 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: https://survstat.rki.de/; Queried on 11-01-2005.

Examples

```
data(hepatitisA)
plot(hepatitisA)
```

`hhh4` *Fitting HHH Models with Random Effects and Neighbourhood Structure*

Description

Fits an autoregressive Poisson or negative binomial model to a univariate or multivariate time series of counts. The characteristic feature of `hhh4` models is the additive decomposition of the conditional mean into *epidemic* and *endemic* components (Held et al, 2005). Log-linear predictors of covariates and random intercepts are allowed in all components; see the Details below. A general introduction to the `hhh4` modelling approach and its implementation is given in `vignette("hhh4")`. Meyer et al (2017, Section 5, available as `vignette("hhh4_spacetime")` describe `hhh4` models for areal time series of infectious disease counts.

Usage

```
hhh4(stsObj,
     control = list(
         ar = list(f = ~ -1, offset = 1, lag = 1),
         ne = list(f = ~ -1, offset = 1, lag = 1,
                   weights = neighbourhood(stsObj) == 1,
                   scale = NULL, normalize = FALSE),
         end = list(f = ~ 1, offset = 1),
         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 = stsObj@epoch - min(stsObj@epoch)),
        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 $y_{i,t-lag}$

`ne` Model for the neighbour-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)`.

scale = NULL optional matrix of the same dimensions as `weights` (or a vector of length `ncol(stsObj)`) to scale the `weights` to `scale * weights`.

normalize = FALSE logical indicating if the (scaled) `weights` should be normalized such that each row sums to 1.

`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)`

`family` Distributional family – either "`Poisson`", or the Negative Binomial distribution. For the latter, the overdispersion parameter can be assumed to be the same for all units ("`NegBin1`"), to vary freely over all units ("`NegBinM`"), or to be shared by some units (specified by a factor of length `ncol(stsObj)`

such that its number of levels determines the number of overdispersion parameters). Note that "NegBinM" is equivalent to factor(colnames(stsObj), levels = colnames(stsObj)).

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 tolerance 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 methods 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. For method="Nelder-Mead", the respective argument is called maxit and defaults to 500.

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 replacing initial values set via fe and ri. Since **surveillance** 1.8-2, named vectors are matched against the coefficient names in the model (where unmatched start values are silently ignored), and need not be complete, e.g., start = list(fixed = c("-log(overdisp)" = 0.5)) (default: 2) for a family = "NegBin1" model. In contrast, an unnamed start vector must specify the full set of parameters as used by the model.

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).

The auxiliary function makeControl might be useful to create such a list of control parameters.

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

An endemic-epidemic multivariate time-series model for infectious disease counts Y_{it} from units $i = 1, \ldots, I$ during periods $t = 1, \ldots, T$ was proposed by Held et al (2005) and was later extended in a series of papers (Paul et al, 2008; Paul and Held, 2011; Held and Paul, 2012; Meyer and Held, 2014). In its most general formulation, this so-called hhh4 (or HHH or H^3 or triple-H) model assumes that, conditional on past observations, Y_{it} has a Poisson or negative binomial distribution with mean

$$\mu_{it} = \lambda_{it} y_{i,t-1} + \phi_{it} \sum_{j \neq i} w_{ji} y_{j,t-1} + e_{it} \nu_{it}$$

In the case of a negative binomial model, the conditional variance is $\mu_{it}(1+\psi_i \mu_{it})$ with overdispersion parameters $\psi_i > 0$ (possibly shared across different units, e.g., $\psi_i \equiv \psi$). Univariate time series of counts Y_t are supported as well, in which case hhh4 can be regarded as an extension of glm.nb to account for autoregression. See the Examples below for a comparison of an endemic-only hhh4 model with a corresponding glm.nb.

The three unknown quantities of the mean μ_{it},

- λ_{it} in the autoregressive (ar) component,
- ϕ_{it} in the neighbour-driven (ne) component, and
- ν_{it} in the endemic (end) component,

are log-linear predictors incorporating time-/unit-specific covariates. They may also contain unit-specific random intercepts as proposed by Paul and Held (2011). The endemic mean is usually modelled proportional to a unit-specific offset e_{it} (e.g., population numbers or fractions); it is possible to include such multiplicative offsets in the epidemic components as well. The w_{ji} are transmission weights reflecting the flow of infections from unit j to unit i. If weights vary over time (prespecified as a 3-dimensional array (w_{jit})), the ne sum in the mean uses $w_{jit} y_{j,t-1}$. In spatial hhh4 applications, the "units" refer to geographical regions and the weights could be derived from movement network data. Alternatively, the weights w_{ji} can be estimated parametrically as a function of adjacency order (Meyer and Held, 2014), see W_powerlaw.

(Penalized) Likelihood inference for such hhh4 models has been established by Paul and Held (2011) with extensions for parametric neighbourhood weights by Meyer and Held (2014). Supplied with the analytical score function and Fisher information, the function hhh4 by default uses the quasi-Newton algorithm available through nlminb to maximize the log-likelihood. Convergence is usually fast even for a large number of parameters. If the model contains random effects, the penalized and marginal log-likelihoods are maximized alternately until convergence.

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)

Michaela Paul, Sebastian Meyer, Leonhard Held

References

Held, L., Höhle, M. and Hofmann, M. (2005): A statistical framework for the analysis of multivariate infectious disease surveillance counts. *Statistical Modelling*, **5** (3), 187-199. doi:10.1191/1471082X05st098oa

Paul, M., Held, L. and Toschke, A. M. (2008): Multivariate modelling of infectious disease surveillance data. *Statistics in Medicine*, **27** (29), 6250-6267. doi:10.1002/sim.4177

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** (10), 1118-1136. doi:10.1002/sim.4177

Held, L. and Paul, M. (2012): Modeling seasonality in space-time infectious disease surveillance data. *Biometrical Journal*, **54** (6), 824-843. doi:10.1002/bimj.201200037

Meyer, S. and Held, L. (2014): Power-law models for infectious disease spread. *The Annals of Applied Statistics*, **8** (3), 1612-1639. doi:10.1214/14AOAS743

Meyer, S., Held, L. and Höhle, M. (2017): Spatio-temporal analysis of epidemic phenomena using the R package **surveillance**. *Journal of Statistical Software*, **77** (11), 1-55. doi:10.18637/jss.v077.i11

See Also

See the special functions fe, ri and the examples below for how to specify unit-specific effects.

Further details on the modelling approach and illustrations of its implementation can be found in vignette("hhh4") and vignette("hhh4_spacetime").

Examples

```
######################
## Univariate examples
######################

### weekly counts of salmonella agona cases, UK, 1990-1995

data("salmonella.agona")
## convert old "disProg" to new "sts" data class
salmonella <- disProg2sts(salmonella.agona)
salmonella
plot(salmonella)

## generate formula for an (endemic) time trend and seasonality
f.end <- addSeason2formula(f = ~1 + t, S = 1, period = 52)
f.end
## specify a simple autoregressive negative binomial model
model1 <- list(ar = list(f = ~1), end = list(f = f.end), family = "NegBin1")
## fit this model to the data
res <- hhh4(salmonella, model1)
## summarize the model fit
summary(res, idx2Exp="ar.1", amplitudeShift=TRUE, maxEV=TRUE)
plot(res)
plot(res, type = "season", components = "end")


### weekly counts of meningococcal infections, Germany, 2001-2006

data("influMen")
fluMen <- disProg2sts(influMen)
meningo <- fluMen[, "meningococcus"]
meningo
plot(meningo)

## again a simple autoregressive NegBin model with endemic seasonality
meningoFit <- hhh4(stsObj = meningo, control = list(
    ar = list(f = ~1),
    end = list(f = addSeason2formula(f = ~1, S = 1, period = 52)),
    family = "NegBin1"
))

summary(meningoFit, idx2Exp=TRUE, amplitudeShift=TRUE, maxEV=TRUE)
plot(meningoFit)
plot(meningoFit, type = "season", components = "end")
```

```
#######################
## Multivariate examples
#######################

### bivariate analysis of influenza and meningococcal infections
### (see Paul et al, 2008)

plot(fluMen, same.scale = FALSE)

## Fit a negative binomial model with
## - autoregressive component: disease-specific intercepts
## - neighbour-driven component: only transmission from flu to men
## - endemic component: S=3 and S=1 sine/cosine pairs for flu and men, respectively
## - disease-specific overdispersion

WfluMen <- neighbourhood(fluMen)
WfluMen["meningococcus","influenza"] <- 0
WfluMen
f.end_fluMen <- addSeason2formula(f = ~ -1 + fe(1, which = c(TRUE, TRUE)),
                                  S = c(3, 1), period = 52)
f.end_fluMen
fluMenFit <- hhh4(fluMen, control = list(
    ar = list(f = ~ -1 + fe(1, unitSpecific = TRUE)),
    ne = list(f = ~ 1, weights = WfluMen),
    end = list(f = f.end_fluMen),
    family = "NegBinM"))
summary(fluMenFit, idx2Exp=1:3)
plot(fluMenFit, type = "season", components = "end", unit = 1)
plot(fluMenFit, type = "season", components = "end", unit = 2)



### weekly counts of measles, Weser-Ems region of Lower Saxony, Germany

data("measlesWeserEms")
measlesWeserEms
plot(measlesWeserEms)  # note the two districts with zero cases

## we could fit the same simple model as for the salmonella cases above
model1 <- list(
    ar = list(f = ~1),
    end = list(f = addSeason2formula(~1 + t, period = 52)),
    family = "NegBin1"
)
measlesFit <- hhh4(measlesWeserEms, model1)
summary(measlesFit, idx2Exp=TRUE, amplitudeShift=TRUE, maxEV=TRUE)

## but we should probably at least use a population offset in the endemic
## component to reflect heterogeneous incidence levels of the districts,
## and account for spatial dependence (here just using first-order adjacency)
measlesFit2 <- update(measlesFit,
    end = list(offset = population(measlesWeserEms)),
    ne = list(f = ~1, weights = neighbourhood(measlesWeserEms) == 1))
```

```
summary(measlesFit2, idx2Exp=TRUE, amplitudeShift=TRUE, maxEV=TRUE)
plot(measlesFit2, units = NULL, hide0s = TRUE)

## 'measlesFit2' corresponds to the 'measlesFit_basic' model in
## vignette("hhh4_spacetime"). See there for further analyses,
## including vaccination coverage as a covariate,
## spatial power-law weights, and random intercepts.


## Not run:
### last but not least, a more sophisticated (and time-consuming)
### analysis of weekly counts of influenza from 140 districts in
### Southern Germany (originally analysed by Paul and Held, 2011,
### and revisited by Held and Paul, 2012, and Meyer and Held, 2014)

data("fluBYBW")
plot(fluBYBW, type = observed ~ time)
plot(fluBYBW, type = observed ~ unit,
     ## mean yearly incidence per 100.000 inhabitants (8 years)
     population = fluBYBW@map$X31_12_01 / 100000 * 8)

## For the full set of models for data("fluBYBW") as analysed by
## Paul and Held (2011), including predictive model assessement
## using proper scoring rules, see the (computer-intensive)
## demo("fluBYBW") script:
demoscript <- system.file("demo", "fluBYBW.R", package = "surveillance")
demoscript
#file.show(demoscript)

## Here we fit the improved power-law model of Meyer and Held (2014)
## - autoregressive component: random intercepts + S = 1 sine/cosine pair
## - neighbour-driven component: random intercepts + S = 1 sine/cosine pair
##   + population gravity with normalized power-law weights
## - endemic component: random intercepts + trend + S = 3 sine/cosine pairs
## - random intercepts are iid but correlated between components
f.S1 <- addSeason2formula(
    ~-1 + ri(type="iid", corr="all"),
    S = 1, period = 52)
f.end.S3 <- addSeason2formula(
    ~-1 + ri(type="iid", corr="all") + I((t-208)/100),
    S = 3, period = 52)

## for power-law weights, we need adjaceny orders, which can be
## computed from the binary adjacency indicator matrix
nbOrder1 <- neighbourhood(fluBYBW)
neighbourhood(fluBYBW) <- nbOrder(nbOrder1)

## full model specification
fluModel <- list(
    ar = list(f = f.S1),
    ne = list(f = update.formula(f.S1, ~ . + log(pop)),
              weights = W_powerlaw(maxlag=max(neighbourhood(fluBYBW)),
                                   normalize = TRUE, log = TRUE)),
```

```
    end = list(f = f.end.S3, offset = population(fluBYBW)),
    family = "NegBin1", data = list(pop = population(fluBYBW)),
    optimizer = list(variance = list(method = "Nelder-Mead")),
    verbose = TRUE)

## CAVE: random effects considerably increase the runtime of model estimation
## (It is usually advantageous to first fit a model with simple intercepts
## to obtain reasonable start values for the other parameters.)
set.seed(1)  # because random intercepts are initialized randomly
fluFit <- hhh4(fluBYBW, fluModel)

summary(fluFit, idx2Exp = TRUE, amplitudeShift = TRUE)

plot(fluFit, type = "fitted", total = TRUE)

plot(fluFit, type = "season")
range(plot(fluFit, type = "maxEV"))

plot(fluFit, type = "maps", prop = TRUE)

gridExtra::grid.arrange(
    grobs = lapply(c("ar", "ne", "end"), function (comp)
        plot(fluFit, type = "ri", component = comp, main = comp,
             exp = TRUE, sub = "multiplicative effect")),
    nrow = 1, ncol = 3)

plot(fluFit, type = "neweights", xlab = "adjacency order")

## End(Not run)


#######################################################################
## An endemic-only "hhh4" model can also be estimated using MASS::glm.nb
#######################################################################

## weekly counts of measles, Weser-Ems region of Lower Saxony, Germany
data("measlesWeserEms")

## fit an endemic-only "hhh4" model
## with time covariates and a district-specific offset
hhh4fit <- hhh4(measlesWeserEms, control = list(
    end = list(f = addSeason2formula(~1 + t, period = frequency(measlesWeserEms)),
               offset = population(measlesWeserEms)),
    ar = list(f = ~-1), ne = list(f = ~-1), family = "NegBin1",
    subset = 1:nrow(measlesWeserEms)
))
summary(hhh4fit)

## fit the same model using MASS::glm.nb
measlesWeserEmsData <- as.data.frame(measlesWeserEms, tidy = TRUE)
measlesWeserEmsData$t <- c(hhh4fit$control$data$t)
glmnbfit <- MASS::glm.nb(
    update(formula(hhh4fit)$end, observed ~ . + offset(log(population))),
```

```
    data = measlesWeserEmsData
)
summary(glmnbfit)

## Note that the overdispersion parameter is parametrized inversely.
## The likelihood and point estimates are all the same.
## However, the variance estimates are different: in glm.nb, the parameters
## are estimated conditional on the overdispersion theta.
```

---

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 random intercept terms, respectively, in the component formulae of `hhh4`. They are not exported and not intended to be called directly.

Usage

```
fe(x, unitSpecific = FALSE, which = NULL, initial = NULL)

ri(type = c("iid","car"), corr = c("none", "all"),
   initial.fe = 0, initial.var = -.5, 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.
`unitSpecific`	logical indicating if the effect of `x` should be unit-specific. This is a convenient shortcut for `which = rep(TRUE, nUnits)`, where `nUnits` is the number of units (i.e., columns of the `"sts"` object).
`which`	vector of logicals indicating which unit(s) should get an unit-specific parameter. For units with a `FALSE` value, the effect term for `x` will be zero in the log-linear predictor. Note especially that setting a `FALSE` value for the intercept term of a unit, e.g., `ar = list(f = ~-1 + fe(1, which=c(TRUE, FALSE)))` in a bivariate `hhh4` model, does *not* mean that the (autoregressive) model component is omitted for this unit, but that $\log(\lambda_1) = \alpha_1$ and $\log(\lambda_2) = 0$, which is usually not of interest. ATM, omitting an autoregressive effect for a specific unit is not possible. If `which=NULL`, the parameter is assumed to be the same for all units.
`initial`	initial values (on internal scale!) for the fixed effects used for optimization. The default (`NULL`) means to use zeroes.
`type`	random intercepts either follow an IID or a CAR model.

`corr`	whether random effects in different components (such as `ar` and `end`) should be correlated or not.
`initial.fe`	initial value for the random intercept mean.
`initial.var`	initial values (on internal scale!) for the variance components used for optimization.
`initial.re`	initial values (on internal scale!) for the random effects used for optimization. The default `NULL` are random numbers from a normal distribution with zero mean and variance 0.001.

Value

a list structure characterizing the term for `hhh4`.

Note

These special functions are intended for use in component formulae of `hhh4` models and are not exported from the package namespace.

If unit-specific fixed or random intercepts are specified, an overall intercept must be excluded (by `-1`) in the component formula.

See Also

`hhh4` model specifications in `vignette("hhh4")`, `vignette("hhh4_spacetime")` or on the help page of `hhh4`.

`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`. The implementation is illustrated in Meyer et al. (2017, Section 5), see `vignette("hhh4_spacetime")`.

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, intercept = FALSE, ...)
## S3 method for class 'hhh4'
coeflist(x, ...)

## 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, ...)

## S3 method for class 'hhh4'
residuals(object, type = c("deviance", "pearson", "response"), ...)
```

Arguments

x, object	an object of class "hhh4".
digits	the number of significant digits to use when printing parameter estimates.
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, ranef, and coeflist 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	a logical value indicating if standard errors are required.
idx2Exp	integer or character vector selecting parameters that should be exponentiated, or NULL for none. The special value idx2Exp = TRUE will exp-transform all parameters except for those associated with log() covariates or already affected by reparamPsi or amplitudeShift.
amplitudeShift	a logical value 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.

`intercept` logical. If `FALSE` (default), the returned random effects represent zero-mean deviations around the corresponding global intercepts of the *log*-linear predictors. Setting `intercept=TRUE` adds these global intercepts to the result (and implies `tomatrix=TRUE`).

`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.

`type` the type of residuals which should be returned. The alternatives are `"deviance"` (default), `"pearson"`, and `"response"`.

Value

The `coef`-method returns all estimated (regression) parameters from a `hhh4` model. If the model includes random effects, those can be extracted with `ranef`, whereas `fixef` returns the fixed parameters. The `coeflist`-method extracts the model coefficients in a list (by parameter group).

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). Note that `idx2Exp` gives (symmetric) Wald intervals for the exponentiated parameters based on the delta method, not `exp(CI)`.

The `residuals`-method extracts raw (`"response"`) or `"deviance"` or standardized (`"pearson"`) residuals from the model fit similar to `residuals.glm` for Poisson or NegBin GLM's. Note that the squared Pearson residual is equivalent to the *normalized squared error score*, which can be computed from the fitted model using `scores(object, "nses")`.

Author(s)

Michaela Paul and Sebastian Meyer

References

Meyer, S., Held, L. and Höhle, M. (2017): Spatio-temporal analysis of epidemic phenomena using the R package **surveillance**. *Journal of Statistical Software*, **77** (11), 1-55. doi:10.18637/jss.v077.i11

See Also

the plot and update methods for fitted "hhh4" models.

---

hhh4_plot *Plots for Fitted* hhh4*-models*

Description

There are six 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, maps of the fitted mean components averaged over time ("maps"), or 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.

Spatio-temporal "hhh4" models and these plots are illustrated in Meyer et al. (2017, Section 5), see vignette("hhh4_spacetime").

Usage

```
## S3 method for class 'hhh4'
plot(x, type=c("fitted", "season", "maxEV", "maps", "ri", "neweights"), ...)

plotHHH4_fitted(x, units = 1, names = NULL,
                col = c("grey85", "blue", "orange"),
                pch = 19, pt.cex = 0.6, pt.col = 1,
                par.settings = list(),
                legend = TRUE, legend.args = list(),
                legend.observed = FALSE,
                decompose = NULL, total = FALSE, meanHHH = NULL, ...)

plotHHH4_fitted1(x, unit = 1, main = NULL,
                 col = c("grey85", "blue", "orange"),
                 pch = 19, pt.cex = 0.6, pt.col = 1, border = col,
                 start = x$stsObj@start, end = NULL, xaxis = NULL,
                 xlim = NULL, ylim = NULL, xlab = "", ylab = "No. infected",
                hide0s = FALSE, decompose = NULL, total = 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, period = NULL)
getMaxEV_season(x, period = frequency(x$stsObj))

plotHHH4_maxEV(...,
               matplot.args = list(), refline.args = list(),
               legend.args = list())
getMaxEV(x)

plotHHH4_maps(x, which = c("mean", "endemic", "epi.own", "epi.neighbours"),
              prop = FALSE, main = which, zmax = NULL,
              col.regions = hcl.colors(10, "Heat 2", rev = TRUE),
              labels = FALSE, sp.layout = NULL, ...,
              map = x$stsObj@map, meanHHH = NULL)

plotHHH4_ri(x, component, exp = FALSE,
            at = list(n = 10), col.regions = hcl.colors(99, "Blue-Red 2"),
            colorkey = TRUE, labels = FALSE, sp.layout = NULL,
            gpar.missing = list(col = "darkgrey", lty = 2, lwd = 2),
            ...)

plotHHH4_neweights(x, plotter = boxplot, ...,
                   exclude = if (isTRUE(x$control$ar$inModel)) 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 `"maps"` of the fitted mean components averaged over time, or a map of estimated region-specific random intercepts (`"ri"`) of a specific model `component`. The latter two require `x$stsObj` to contain a map.
`...`	For `plotHHH4_season` and `plotHHH4_maxEV`, one or more `hhh4`-fits, or a single list of these. Otherwise further arguments passed on to other functions. For the `plot`-method these go to the specific plot `type` function. `plotHHH4_fitted` passes them to `plotHHH4_fitted1`, which is called sequentially for every unit in `units`. `plotHHH4_maps` and `plotHHH4_ri` pass additional arguments to `spplot`, and `plotHHH4_neweights` to the `plotter`.
`units`, `unit`	integer or character vector specifying a single `unit` or possibly multiple `units` to plot. It indexes `colnames(x$stsObj)`. In `plotHHH4_fitted`, `units=NULL` plots all units.

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 3 vectors specifying the fill and border colors for the endemic, autoregressive, and spatio-temporal component polygons (in this order).

`pch`, `pt.cex`, `pt.col`
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.

`decompose` if `TRUE` or (a permutation of) `colnames(x$stsObj)`, the fitted mean will be decomposed into the contributions from each single unit and the endemic part instead of the default endemic + AR + neighbours decomposition.

`total` logical indicating if the fitted components should be summed over all units to be compared with the total observed counts at each time point. If `total=TRUE`, the `units`/`unit` argument is ignored.

`start`, `end` time range to plot specified by vectors of length two in the form `c(year,number)`, see "`sts`".

`xaxis` if this is a list (of arguments for `addFormattedXAxis`), the time axis is nicely labelled similar to `stsplot_time`. Note that in this case or if `xaxis = NA`, the basic time indexes `1:nrow(x$stsObj)` will be used as x coordinates, which is different from the long-standing default (`xaxis = NULL`) with a real time scale.

`xlim` numeric vector of length 2 specifying the x-axis range. The default (`NULL`) is to plot the complete time range (`type="fitted"`) or period (`type="season"`), respectively.

`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 include the global intercept. For `plotHHH4_season`, the default (`FALSE`) means to plot seasonality as a multiplicative effect on the respective component. Multiplication by the intercept only makes sense if there are no further (non-centered) covariates/offsets in the component.

`exp` logical indicating whether to `exp`-transform the color-key axis labels to show the multiplicative effect of the region-specific random intercept on the respective component. Axis labels are then computed using `log_breaks` from package **scales** (if that is available) or `axisTicks` (as a fallback) respecting the `colorkey$tick.number` setting (default: 7). The default is `FALSE`.

`at` a numeric vector of breaks for the color levels (see `levelplot`), or a list specifying the number of breaks `n` (default: 10) and their `range` (default: range of the random effects, extended to be symmetric around 0). In the latter case, breaks are equally spaced (on the original, non-`exp` scale of the random intercepts). If `exp=TRUE`, custom breaks (or `range`) need to be given on the exp-scale.

`matplot.args` list of line style specifications passed to `matplot`, e.g., `lty`, `lwd`, `col`.

`refline.args` list of line style specifications (e.g., `lty` or `col`) passed to `abline` when drawing the reference line (`h=1`) in plots of seasonal effects (if `intercept=FALSE`) and of the dominant eigenvalue. The reference line is omitted if `refline.args` is not a list.

`period` a numeric value giving the (longest) period of the harmonic terms in the model. This usually coincides with the `freq` of the data (the default), but needs to be adjusted if the model contains harmonics with a longer periodicity.

`which` a character vector specifying the components of the mean for which to produce maps. By default, the overall mean and all three components are shown.

`prop` a logical indicating whether the component maps should display proportions of the total mean instead of absolute numbers.

`zmax` a numeric vector of length `length(which)` (recycled as necessary) specifying upper limits for the color keys of the maps, using a lower limit of 0. A missing element (`NA`) means to use a map-specific color key only covering the range of the values in that map (can be useful for `prop = TRUE`). The default `zmax = NULL` means to use the same scale for the component maps and a separate scale for the map showing the overall mean.

`col.regions` a vector of colors used to encode the fitted component means (see `levelplot`). For `plotHHH4_maps`, the length of this color vector also determines the number of levels (10 by default).

colorkey	a Boolean indicating whether to draw the color key. Alternatively, a list specifying how to draw it, see levelplot.
map	an object inheriting from "SpatialPolygons" with row.names covering colnames(x).
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 for sp.polygons, applied to regions with missing random intercepts, i.e., not included in the model. Such extra regions won't be plotted 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 excluded if the model contains an AR component (when it will usually be zero).
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="maxEV"), "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_maps returns a trellis.object if length(which) == 1 (a single spplot), and otherwise uses grid.arrange from the **gridExtra** package to arrange all length(which) spplots on a single page. plotHHH4_ri returns the generated spplot, i.e., a trellis.object.
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:10.1002/bimj.201200037

Meyer, S., Held, L. and Höhle, M. (2017): Spatio-temporal analysis of epidemic phenomena using the R package **surveillance**. *Journal of Statistical Software*, **77** (11), 1-55. doi:10.18637/jss.v077.i11

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)

## sum fitted components over all units
plot(measlesFit, total=TRUE)

## 'xaxis' option for a nicely formatted time axis
## default tick locations and labels:
plot(measlesFit, total=TRUE, xaxis=list(epochsAsDate=TRUE, line=1))
## an alternative with monthly ticks:
oopts <- surveillance.options(stsTickFactors = c("%m"=0.75, "%Y" = 1.5))
plot(measlesFit, total=TRUE, xaxis=list(epochsAsDate=TRUE,
    xaxis.tickFreq=list("%m"=atChange, "%Y"=atChange),
    xaxis.labelFreq=list("%Y"=atMedian), xaxis.labelFormat="%Y"))
surveillance.options(oopts)

## plot the multiplicative effect of seasonality
plot(measlesFit, type="season")

## alternative fit with biennial pattern, plotted jointly with original fit
measlesFit2 <- update(measlesFit,
    end = list(f = addSeason2formula(~0 + ri(type="iid"), S=2, period=104)))
plotHHH4_season(measlesFit, measlesFit2, components="end", period=104)

## 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

## fitted mean components/proportions by district, averaged over time
if (requireNamespace("gridExtra")) {
    plot(measlesFit, type="maps", labels=list(cex=0.6),
```

```
        which=c("endemic", "epi.own"), prop=TRUE, zmax=NA,
        main=c("endemic proportion", "autoregressive proportion"))
}

## estimated random intercepts of the endemic component
round(nu0    <- fixef(measlesFit)["end.ri(iid)"],   4) # global intercept
round(ranefs <- ranef(measlesFit, tomatrix = TRUE), 4) # zero-mean deviations
stopifnot(all.equal(
    nu0 + ranefs,
    ranef(measlesFit, intercept = TRUE) # local intercepts (log-scale)
))
plot(measlesFit, type="ri", component="end",
     main="deviations around the endemic intercept (log-scale)")
exp(ranef(measlesFit))  # multiplicative effects, plotted below
plot(measlesFit, type="ri", component="end", exp=TRUE,
     main="multiplicative effects",
     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"))
```

`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 autoregressive 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.

Note

Predictions for "newdata", i.e., with modified covariates or fixed weights, can be computed manually by adjusting the control list (in a copy of the original fit), dropping the old terms, and using the internal function meanHHH directly, see the Example.

Author(s)

Michaela Paul and Sebastian Meyer

Examples

```
## simulate simple seasonal noise with reduced baseline for t >= 60
t <- 0:100
y <- rpois(length(t), exp(3 + sin(2*pi*t/52) - 2*(t >= 60)))
obj <- sts(y)
plot(obj)

## fit true model
fit <- hhh4(obj, list(end = list(f = addSeason2formula(~lock)),
                      data = list(lock = as.integer(t >= 60)),
                      family = "Poisson"))
coef(fit, amplitudeShift = TRUE, se = TRUE)

## compute predictions for a subset of the time points
stopifnot(identical(predict(fit), fitted(fit)))
plot(obj)
lines(40:80, predict(fit, newSubset = 40:80), lwd = 2)


## advanced: compute predictions for "newdata" (here, a modified covariate)
mod <- fit
mod$terms <- NULL  # to be sure
mod$control$data$lock[t >= 60] <- 0.5
pred <- meanHHH(mod$coefficients, terms(mod))$mean
plot(fit, xaxis = NA)
lines(mod$control$subset, pred, lty = 2)
```

hhh4_simulate *Simulate "hhh4" Count Time Series*

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 (and cannot exceed) the whole period defined by the underlying "sts" object.
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), which especially means reparamPsi = TRUE (as per default when using the coef-method to extract the parameters). The overdispersion parameter in coefs is the inverse of the dispersion parameter size in rnbinom.
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 object of class "`hhh4sims`", which is an array of dimension `c(length(subset), ncol(object$stsObj), nsim)`. The originally observed counts during the simulation period, `object$stsObj[subset,]`, are attached for reference (used by the `plot`-methods) as an attribute "`stsObserved`", and the initial condition `y.start` as attribute "`initial`". The `[`-method for "`hhh4sims`" takes care of subsetting these attributes appropriately.

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

`plot.hhh4sims` and `scores.hhh4sims` and the examples therein for `nsim > 1`.

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(~1, period = 52)),
              family = "NegBin1"))
plot(fit)

# simulate from model (generates an "sts" object)
simData <- simulate(fit, seed=1234)

# plot simulated data
```

```
plot(simData, main = "simulated data", xaxis.labelFormat=NULL)

# use simplify=TRUE to return an array of simulated counts
simCounts <- simulate(fit, seed=1234, simplify=TRUE)
dim(simCounts)  # nTime x nUnit x nsim

# plot the first year of simulated counts (+ initial + observed)
plot(simCounts[1:52,,], type = "time", xaxis.labelFormat = NULL)
# see help(plot.hhh4sims) for other plots, mainly useful for nsim > 1

# simulate from a Poisson instead of a NegBin model
# keeping all other parameters fixed at their original estimates
coefs <- replace(coef(fit), "overdisp", 0)
simData2 <- simulate(fit, seed=123, coefs = coefs)
plot(simData2, main = "simulated data: Poisson model", xaxis.labelFormat = NULL)

# simulate from a model with higher autoregressive parameter
coefs <- replace(coef(fit), "ar.1", log(0.9))
simData3 <- simulate(fit, seed=321, coefs = coefs)
plot(simData3, main = "simulated data: lambda = 0.5", xaxis.labelFormat = NULL)


## more sophisticated: simulate beyond initially observed time range

# extend data range by one year (non-observed domain), filling with NA values
nextend <- 52
timeslots <- c("observed", "state", "alarm", "upperbound", "populationFrac")
addrows <- function (mat, n) mat[c(seq_len(nrow(mat)), rep(NA, n)),,drop=FALSE]
extended <- Map(function (x) addrows(slot(meningo, x), n = nextend), x = timeslots)
# create new sts object with extended matrices
meningo2 <- do.call("sts", c(list(start = meningo@start, frequency = meningo@freq,
                                  map = meningo@map), extended))

# fit to the observed time range only, via the 'subset' argument
fit2 <- hhh4(meningo2, control = list(
              ar = list(f = ~ 1),
              end = list(f = addSeason2formula(~1, period = 52)),
              family = "NegBin1",
              subset = 2:(nrow(meningo2) - nextend)))
# the result is the same as before
stopifnot(all.equal(fit, fit2, ignore = c("stsObj", "control")))

# long-term probabilistic forecast via simulation for non-observed time points
meningoSim <- simulate(fit2, nsim = 100, seed = 1,
                       subset = seq(nrow(meningo)+1, nrow(meningo2)),
                       y.start = tail(observed(meningo), 1))
apply(meningoSim, 1:2, function (ysim) quantile(ysim, c(0.1, 0.5, 0.9)))
# three plot types are available for "hhh4sims", see also ?plot.hhh4sims
plot(meningoSim, type = "time", average = median)
plot(meningoSim, type = "size", observed = FALSE)
if (requireNamespace("fanplot"))
    plot(meningoSim, type = "fan", means.args = list(),
         fan.args = list(ln = c(.1,.9), ln.col = 8))
```

hhh4_simulate_plot *Plot Simulations from "hhh4" Models*

Description

Arrays of simulated counts from simulate.hhh4 can be visualized as final size boxplots, individual or average time series, or fan charts (using the **fanplot** package). An aggregate-method is also available.

Usage

```
## S3 method for class 'hhh4sims'
plot(x, ...)
## S3 method for class 'hhh4sims'
aggregate(x, units = TRUE, time = FALSE, ..., drop = FALSE)

as.hhh4simslist(x, ...)
## S3 method for class 'hhh4simslist'
plot(x, type = c("size", "time", "fan"), ...,
     groups = NULL, par.settings = list())
## S3 method for class 'hhh4simslist'
aggregate(x, units = TRUE, time = FALSE, ..., drop = FALSE)

plotHHH4sims_size(x, horizontal = TRUE, trafo = NULL, observed = TRUE,
                  names = base::names(x), ...)

plotHHH4sims_time(x, average = mean, individual = length(x) == 1,
    conf.level = if (individual) 0.95 else NULL,
    matplot.args = list(), initial.args = list(), legend = length(x) > 1,
    xlim = NULL, ylim = NULL, add = FALSE, ...)

plotHHH4sims_fan(x, which = 1,
    fan.args = list(), observed.args = list(), initial.args = list(),
    means.args = NULL, key.args = NULL, xlim = NULL, ylim = NULL,
    add = FALSE, xaxis = list(), ...)
```

Arguments

x	an object of class "hhh4sims" (as resulting from the simulate-method for "hhh4" models if simplify = TRUE was set), or an "hhh4simslist", i.e., a list of such simulations potentially obtained from different model fits (using the same simulation period).
type	a character string indicating the summary plot to produce.
...	further arguments passed to methods.
groups	an optional factor to produce stratified plots by groups of units. The special setting groups = TRUE is a convenient shortcut for one plot by unit.

par.settings	a list of graphical parameters for par. Sensible defaults for mfrow, mar and las will be applied unless overridden or !is.list(par.settings).
horizontal	a logical indicating if the boxplots of the final size distributions should be horizontal (the default).
trafo	an optional transformation function from the **scales** package, e.g., sqrt_trans.
observed	a logical indicating if a line and axis value for the observed size of the epidemic should be added to the plot. Alternatively, a list with graphical parameters can be specified to modify the default values.
names	a character vector of names for x.
average	scalar-valued function to apply to the simulated counts at each time point.
individual	a logical indicating if the individual simulations should be shown as well.
conf.level	a scalar in (0,1), which determines the level of the pointwise quantiles obtained from the simulated counts at each time point. A value of NULL disables the confidence interval.
matplot.args	a list of graphical parameters for matlines.
initial.args	if a list (of graphical parameters for lines), a bar for the initial number of cases is added to the plot.
legend	a logical, a character vector (providing names for x), or a list of parameters for legend.
xlim, ylim	vectors of length 2 determining the axis limits.
add	a logical indicating if the (mean) simulated time series or the fan chart, respectively, should be added to an existing plot.
which	a single integer or a character string selecting the model in x for which to produce the fan chart. This is only relevant if x is a "hhh4simslist" of simulations from multiple models. Defaults to the first model.
fan.args	a list of graphical parameters for the fan, e.g., to employ a different colorRampPalette as fan.col, or to enable contour lines via ln.
observed.args	if a list (of graphical parameters for lines), the originally observed counts are added to the plot.
means.args	if a list (of graphical parameters for lines), the point forecasts are added to the plot (by default as a white line within the fan).
key.args	if a list, a color key (in fan's "boxfan"-style) is added to the fan chart. The list may include positioning parameters start (the x-position) and ylim (the y-range of the color key), space to modify the width of the boxfan, and rlab to modify the labels. The color key is disabled by default. An alternative way of labeling the quantiles is via the argument ln in fan.args, see the Examples.
xaxis	if a list of arguments for addFormattedXAxis, that function is used to draw the time axis, otherwise a default x-axis is drawn.
units	a logical indicating aggregation over units. Can also be a factor (or something convertible to a factor using as.factor) to aggregate groups of units.
time	a logical indicating if the counts should be summed over the whole simulation period.
drop	a logical indicating if the unit dimension and the "hhh4sims" (or "hhh4simslist") class should be dropped after aggregating over (groups of) units.

Author(s)

Sebastian Meyer

Examples

```
### univariate example
data("salmAllOnset")

## fit a hhh4 model to the first 13 years
salmModel <- list(end = list(f = addSeason2formula(~1 + t)),
                  ar = list(f = ~1), family = "NegBin1", subset = 2:678)
salmFit <- hhh4(salmAllOnset, salmModel)

## simulate the next 20 weeks ahead
salmSims <- simulate(salmFit, nsim = 300, seed = 3, subset = 678 + seq_len(20),
                     y.start = observed(salmAllOnset)[678,])

## compare final size distribution to observed value
summary(aggregate(salmSims, time = TRUE))  # summary of simulated values
plot(salmSims, type = "size")

## individual and average simulated time series with a confidence interval
plot(salmSims, type = "time", main = "20-weeks-ahead simulation")

## fan chart based on the quantiles of the simulated counts at each time point
## point forecasts are represented by a white line within the fan
if (requireNamespace("fanplot")) {
    plot(salmSims, type = "fan", main = "20-weeks-ahead simulation",
         fan.args = list(ln = 1:9/10), means.args = list())
}


### multivariate example
data("measlesWeserEms")

## fit a hhh4 model to the first year
measlesModel <- list(
    end = list(f = addSeason2formula(~1), offset = population(measlesWeserEms)),
    ar = list(f = ~1),
    ne = list(f = ~1 + log(pop),
        weights = W_powerlaw(maxlag = 5, normalize = TRUE)),
    family = "NegBin1", subset = 2:52,
    data = list(pop = population(measlesWeserEms)))
measlesFit1 <- hhh4(measlesWeserEms, control = measlesModel)

## use a Poisson distribution instead (just for comparison)
measlesFit2 <- update(measlesFit1, family = "Poisson")

## simulate realizations from these models during the second year
measlesSims <- lapply(X = list(NegBin = measlesFit1, Poisson = measlesFit2),
                      FUN = simulate, nsim = 50, seed = 1, subset = 53:104,
                      y.start = observed(measlesWeserEms)[52,])
```

```
## final size of the first model
plot(measlesSims[[1]])

## stratified by groups of districts
mygroups <- factor(substr(colnames(measlesWeserEms), 4, 4))
apply(aggregate(measlesSims[[1]], time = TRUE, units = mygroups), 1, summary)
plot(measlesSims[[1]], groups = mygroups)

## a class and plot-method for a list of simulations from different models
measlesSims <- as.hhh4simslist(measlesSims)
plot(measlesSims)

## simulated time series
plot(measlesSims, type = "time", individual = TRUE, ylim = c(0, 80))

## fan charts
if (requireNamespace("fanplot")) {
    opar <- par(mfrow = c(2,1))
    plot(measlesSims, type = "fan", which = 1, ylim = c(0, 80), main = "NegBin",
         key.args = list())
    plot(measlesSims, type = "fan", which = 2, ylim = c(0, 80), main = "Poisson")
    par(opar)
}
```

---

hhh4_simulate_scores *Proper Scoring Rules for Simulations from* hhh4 *Models*

Description

Calculate proper scoring rules based on simulated predictive distributions.

Usage

```
## S3 method for class 'hhh4sims'
scores(x, which = "rps", units = NULL, ..., drop = TRUE)
## S3 method for class 'hhh4simslist'
scores(x, ...)
```

Arguments

x	an object of class "hhh4sims" (as resulting from the simulate-method for "hhh4" models if simplify = TRUE was set), or an "hhh4simslist", i.e., a list of such simulations potentially obtained from different model fits (using the same simulation period).
which	a character vector indicating which proper scoring rules to compute. By default, only the ranked probability score ("rps") is calculated. Other options include "logs" and "dss".

units	if non-NULL, an integer or character vector indexing the columns of x for which to compute the scores.
drop	a logical indicating if univariate dimensions should be dropped (the default).
...	unused (argument of the generic).

Details

This implementation can only compute *univariate scores*, i.e., independently for each time point.

The logarithmic score is badly estimated if the domain is large and there are not enough samples to cover the underlying distribution in enough detail (the score becomes infinite when an observed value does not occur in the samples). An alternative is to use kernel density estimation as implemented in the R package **scoringRules**.

Value

a numeric vector (for a univariate model, single score, and drop=TRUE) or a numeric array (where the last index corresponds to the scores selected by which).

Author(s)

Sebastian Meyer

Examples

```
data("salmAllOnset")

## fit a hhh4 model to the first 13 years
salmModel <- list(end = list(f = addSeason2formula(~1 + t)),
                  ar = list(f = ~1), family = "NegBin1", subset = 2:678)
salmFit <- hhh4(salmAllOnset, salmModel)

## simulate the next 20 weeks ahead (with very small 'nsim' for speed)
salmSims <- simulate(salmFit, nsim = 500, seed = 3, subset = 678 + seq_len(20),
                     y.start = observed(salmAllOnset)[678,])
if (requireNamespace("fanplot"))
    plot(salmSims, "fan")


### calculate scores at each time point

## using empirical distribution of simulated counts as forecast distribution
scores(salmSims, which = c("rps", "logs", "dss"))
## observed count sometimes not covered by simulations -> infinite log-score
## => for a more detailed forecast, either considerably increase 'nsim', or:

## 1. use continuous density() of simulated counts as forecast distribution
fi <- apply(salmSims, 1, function (x) approxfun(density(x)))
logs_kde <- mapply(function (f, y) -log(f(y)),
                   f = fi, y = observed(attr(salmSims,"stsObserved")))
cbind("empirical" = scores(salmSims, "logs"), "density" = logs_kde)
## a similar KDE approach is implemented in scoringRules::logs_sample()
```

```
## 2. average conditional predictive NegBin's of simulated trajectories,
##    currently only implemented in HIDDA.forecasting::dhhh4sims()


### produce a PIT histogram

## using empirical distribution of simulated counts as forecast distribition
pit(x = observed(attr(salmSims, "stsObserved")),
    pdistr = apply(salmSims, 1:2, ecdf))
## long-term forecast is badly calibrated (lower tail is unused, see fan above)
## we also get a warning for the same reason as infinite log-scores
```

---

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 = object$convergence, evaluate = TRUE)
```

Arguments

object	a fitted "hhh4" model. Non-convergent fits can be updated as well.
...	components modifying the original control list for hhh4. Modifications are performed by modifyList(object$control, list(...)).
S	a named list of numeric vectors serving as argument for addSeason2formula, or NULL (meaning no modification of seasonal terms). This argument provides a convenient way of changing the number of harmonics in the formulae of the model components "ar", "ne" and "end" (to be used as names of the list). Non-specified components are not touched. Updating the formula of component *comp* works by first dropping all sine and cosine terms and then applying addSeason2formula with arguments S = S[[*comp*]] and period = frequency(object$stsObj), unless the component was originally disabled (f = ~ -1) when the harmonics are added to a simple intercept model and a warning is given. Note that this step of updating seasonality is processed after modification of the control list by the ... arguments.
subset.upper	if a scalar value, refit the model to the data up to the time index given by subset.upper. The lower time index remains unchanged, i.e., control$subset[1]:subset.upper is used as the new subset. This argument is used by oneStepAhead.

use.estimates logical specifying if `coef(object)` should be used as starting values for the new fit (which is the new default since **surveillance** 1.8-2, in case the original fit has converged). This works by matching names against the coefficients of the new model. Extra coefficients no longer in the model are silently ignored. Setting `use.estimates = FALSE` means to re-use the previous start specification `object$control$start`.
Note that coefficients can also receive initial values from an extra `start` argument in the update call (as in hhh4), which then takes precedence over `coef(object)`.

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(~1))))

## the same, updating the minimal endemic-only model via 'S' (with a warning):
fit0.2 <- update(hhh4(salmonella), # has no AR component
                 S = list(ar = 0, end = 1))
local({
  fit0$control$start <- fit0.2$control$start <- NULL # obviously different
  stopifnot(all.equal(fit0, fit0.2))
})

## multiple updates: Poisson -> NegBin1, more harmonics
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. These can be inspected using the quantile, confint or plot methods. The associated scores-method computes a number of (strictly) proper scoring rules based on such one-step-ahead predictions; see Paul and Held (2011) for details. There are also calibrationTest and pit methods for oneStepAhead predictions.

Scores, calibration tests and PIT histograms can also be computed for the fitted values of an hhh4 model (i.e., in-sample/training data evaluation).

Usage

```
oneStepAhead(result, tp, type = c("rolling", "first", "final"),
             which.start = c("current", "final"),
             keep.estimates = FALSE, verbose = type != "final",
             cores = 1)

## S3 method for class 'oneStepAhead'
quantile(x, probs = c(2.5, 10, 50, 90, 97.5)/100, ...)
## S3 method for class 'oneStepAhead'
confint(object, parm, level = 0.95, ...)
## S3 method for class 'oneStepAhead'
plot(x, unit = 1, probs = 1:99/100,
     start = NULL, means.args = NULL, ...)

## assessment of "oneStepAhead" predictions
## S3 method for class 'oneStepAhead'
scores(x, which = c("logs", "rps", "dss", "ses"),
       units = NULL, sign = FALSE, individual = FALSE, reverse = FALSE, ...)
## S3 method for class 'oneStepAhead'
calibrationTest(x, units = NULL, ...)
## S3 method for class 'oneStepAhead'
pit(x, units = NULL, ...)

## assessment of the "hhh4" model fit (in-sample predictions)
## S3 method for class 'hhh4'
scores(x, which = c("logs", "rps", "dss", "ses"),
       subset = x$control$subset, units = seq_len(x$nUnit), sign = FALSE, ...)
## S3 method for class 'hhh4'
calibrationTest(x,
                subset = x$control$subset, units = seq_len(x$nUnit), ...)
## S3 method for class 'hhh4'
pit(x, subset = x$control$subset, units = seq_len(x$nUnit), ...)
```

Arguments

result	fitted hhh4 model (class "hhh4").
tp	numeric vector of length 2 specifying the time range in which to compute one-step-ahead predictions (for the time points tp[1]+1, ..., tp[2]+1). If a single time index is specified, it is interpreted as tp[1], and tp[2] is set 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$subset.
which.start	Which initial parameter values should be used when successively refitting the model to subsets of the data (up to time point tp[1], up to tp[1]+1, ...) if type="rolling"? Default ("current") is to use the parameter estimates from the previous time point, and "final" means to always use the estimates from result as initial values. Alternatively, which.start can be a list of start values as expected by hhh4, which then replace the corresponding estimates from result as initial values. This argument is ignored for "non-rolling" types.
keep.estimates	logical indicating if parameter estimates and log-likelihoods from the successive fits should be returned.
verbose	non-negative integer (usually in the range 0:3) specifying the amount of tracing information to output. During hhh4 model updates, the following verbosity is used: 0 if cores > 1, otherwise verbose-1 if there is more than one time point to predict, otherwise verbose.
cores	the number of cores to use when computing the predictions for the set of time points tp in parallel (with mclapply). Note that parallelization is not possible in the default setting type="rolling" and which.start="current" (use which.start="final" for this to work).
object	an object of class "oneStepAhead".
parm	unused (argument of the generic).
level	required confidence level of the prediction interval.
probs	numeric vector of probabilities with values in [0,1].
unit	single integer or character selecting a unit for which to produce the plot.
start	x-coordinate of the first prediction. If start=NULL (default), this is derived from x.
means.args	if a list (of graphical parameters for lines), the point predictions (from x$pred) are added to the plot.
x	an object of class "oneStepAhead" or "hhh4".
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 hence not computed by default.
It is possible to name own scoring rules in which. These must be functions of (x, mu, size), vectorized in all arguments (time x unit matrices) except that size is NULL in case of a Poisson model. See the available scoring rules for guidance, e.g., dss.

subset — subset of time points for which to calculate the scores (or test calibration, or produce the PIT histogram, respectively). Defaults to the subset used for fitting the model.

units — integer or character vector indexing the units for which to compute the scores (or the calibration test or the PIT histogram, respectively). By default, 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.

reverse — logical indicating if the rows (time points) should be reversed in the result. The long-standing but awkward default was to do so for the oneStepAhead-method. This has changed in version 1.16.0, so time points are no longer reversed by default.

... — Unused by the quantile, confint and scores methods.
The plot-method passes further arguments to the fanplot function, e.g., fan.args, observed.args, and key.args can be used to modify the plotting style.
For the calibrationTest-method, further arguments are passed to calibrationTest.default, e.g., which to select a scoring rule.
For the pit-methods, further arguments are passed to pit.default.

Value

oneStepAhead returns a list (of class "oneStepAhead") 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" or shared overdispersion). 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 `quantile`-method computes quantiles of the one-step-ahead forecasts. If there is only one unit, it returns a tp x prob matrix, otherwise a tp x unit x prob array. The `confint`-method is a convenient wrapper with `probs` set according to the required confidence level.

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.

The `calibrationTest`- and `pit`-methods are just convenient wrappers around the respective default methods.

Author(s)

Sebastian Meyer and Michaela Paul

References

Czado, C., Gneiting, T. and Held, L. (2009): Predictive model assessment for count data. *Biometrics*, **65** (4), 1254-1261. doi:10.1111/j.15410420.2009.01191.x

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** (10), 1118-1136. doi:10.1002/sim.4177

See Also

`vignette("hhh4")` and `vignette("hhh4_spacetime")`

Examples

```
### univariate salmonella agona count time series

data("salmonella.agona")
## convert from old "disProg" to new "sts" class
salmonella <- disProg2sts(salmonella.agona)

## generate formula for temporal and seasonal trends
f.end <- addSeason2formula(~1 + t, S=1, period=52)
model <- list(ar = list(f = ~1), end = list(f = f.end), family = "NegBin1")
## fit the model
result <- hhh4(salmonella, model)

## do sequential one-step-ahead predictions for the last 5 weeks
pred <- oneStepAhead(result, nrow(salmonella)-5, type="rolling",
                     which.start="final", verbose=FALSE)
```

```
pred
quantile(pred)
confint(pred)

## simple plot of the 80% one-week-ahead prediction interval
## and point forecasts
if (requireNamespace("fanplot"))
    plot(pred, probs = c(.1,.9), means.args = list())



## note: oneStepAhead(..., type="final") just means fitted values
stopifnot(identical(
    unname(oneStepAhead(result, nrow(salmonella)-5, type="final")$pred),
    unname(tail(fitted(result), 5))))


## compute scores of the one-step-ahead predictions
(sc <- scores(pred))

## the above uses the scores-method for "oneStepAhead" predictions,
## which is a simple wrapper around the default method:
scores(x = pred$observed, mu = pred$pred, size = exp(pred$psi))

## scores with respect to the fitted values are similar
(scFitted <- scores(result, subset = nrow(salmonella)-(4:0)))




## test if the one-step-ahead predictions are calibrated
calibrationTest(pred)  # p = 0.8746

## the above uses the calibrationTest-method for "oneStepAhead" predictions,
## which is a simple wrapper around the default method:
calibrationTest(x = pred$observed, mu = pred$pred, size = exp(pred$psi))

## we can also test calibration of the fitted values
## using the calibrationTest-method for "hhh4" fits
calibrationTest(result, subset = nrow(salmonella)-(4:0))


## plot a (non-randomized) PIT histogram for the predictions
pit(pred)

## the above uses the pit-method for "oneStepAhead" predictions,
## which is a simple wrapper around the default method:
pit(x = pred$observed, pdistr = "pnbinom", mu = pred$pred, size = exp(pred$psi))


### multivariate measles count time series
## (omitting oneStepAhead forecasts here to keep runtime low)
```

```
data("measlesWeserEms")

## simple hhh4 model with random effects in the endemic component
measlesModel <- list(
    end = list(f = addSeason2formula(~0 + ri(type="iid"))),
    ar = list(f = ~1),
    family = "NegBin1")
measlesFit <- hhh4(measlesWeserEms, control = measlesModel)

## assess overall (in-sample) calibration of the model, i.e.,
## if the observed counts are from the fitted NegBin distribution
calibrationTest(measlesFit) # default is DSS (not suitable for low counts)
calibrationTest(measlesFit, which = "logs") # p = 0.7238

## to assess calibration in the second year for a specific district
calibrationTest(measlesFit, subset = 53:104, units = "03452", which = "rps")
pit(measlesFit, subset = 53:104, units = "03452")


### For a more sophisticated multivariate analysis of
### areal time series of influenza counts - data("fluBYBW") -
### see the (computer-intensive) demo("fluBYBW") script:
demoscript <- system.file("demo", "fluBYBW.R", package = "surveillance")
#file.show(demoscript)
```

---

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 $w_{ji} = o_{ji}^{-d}$ (if $o_{ji} > 0$, otherwise $w_{ji} = 0$), where o_{ji} $(= o_{ij})$ is the adjacency order between regions i and j, and the decay parameter d is to be estimated. In the nonparametric formulation, unconstrained log-weights will be estimated for each of the adjacency orders `2:maxlag` (the first-order weight is fixed to 1 for identifiability). Both weight functions can be modified to include a 0-distance weight, which enables hhh4 models without a separate autoregressive component.

Usage

```
W_powerlaw(maxlag, normalize = TRUE, log = FALSE,
           initial = if (log) 0 else 1, from0 = FALSE)

W_np(maxlag, truncate = TRUE, normalize = TRUE,
     initial = log(zetaweights(2:(maxlag+from0))),
     from0 = FALSE, to0 = truncate)
```

Arguments

`maxlag`	a single integer specifying a limiting order of adjacency. If spatial dependence is not to be truncated at some high order, `maxlag` should be set to the maximum adjacency order in the network of regions. The smallest possible value for `maxlag` is 2 if `from0=FALSE` and 1 otherwise.
`truncate, to0`	`W_np` represents order-specific log-weights up to order `maxlag`. Higher orders are by default (`truncate=TRUE`) assumed to have zero weight (similar to `W_powerlaw`). Alternatively, `truncate=FALSE` requests that the weight at order `maxlag` should be carried forward to higher orders. `truncate` has previously been called `to0` (deprecated).
`normalize`	logical indicating if the weights should be normalized such that the rows of the weight matrix sum to 1 (default). Note that normalization does not work with islands, i.e., regions without neighbours.
`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.
`from0`	logical indicating if these parametric weights should include the 0-distance (autoregressive) case. In the default setting (`from0 = FALSE`), adjacency order 0 has zero weight, which is suitable for `hhh4` models with a separate autoregressive component. With `from0 = TRUE` (Meyer and Held, 2017), the power law is based on $(o_{ji} + 1)$, and nonparametric weights are estimated for adjacency orders `1:maxlag`, respectively, where the 0-distance weight is $w_{jj} = 1$ (without normalization). Note that the corresponding `hhh4` model should then exclude a separate autoregressive component (`control$ar$f = ~ -1`).

Details

`hhh4` will take adjacency orders from the `neighbourhood` slot of the `"sts"` object, so these must be prepared before fitting a model with parametric neighbourhood weights. The function `nbOrder` can be used to derive adjacency orders from a binary adjacency matrix.

Value

a list which can be passed as a specification of parametric neighbourhood weights in the `control$ne$weights` 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:10.1214/14AOAS743

Meyer, S. and Held, L. (2017): Incorporating social contact data in spatio-temporal models for infectious disease spread. *Biostatistics*, **18** (2), 338-351. doi:10.1093/biostatistics/kxw051

See Also

nbOrder to determine adjacency orders from a binary adjacency matrix.

getNEweights and coefW to extract the estimated neighbourhood weight matrix and coefficients from an hhh4 model.

Examples

```
data("measlesWeserEms")

## data contains adjaceny orders as required for parametric weights
plot(measlesWeserEms, ~unit, labels = TRUE)
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)),
    end = list(f = addSeason2formula(~-1 + ri(), S=1, period=52)),
    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
coefW(measlesFit)

## plot the spatio-temporal weights o_ji^-d / sum_k o_jk^-d
## as a function of adjacency order
plot(measlesFit, type = "neweights", xlab = "adjacency order")
## normalization => same distance does not necessarily mean same weight.
## to extract the whole weight matrix W: getNEweights(measlesFit)

## visualize contributions of the three model components
## to the overall number of infections (aggregated over all districts)
plot(measlesFit, total = TRUE)
## little contribution from neighbouring districts


if (surveillance.options("allExamples")) {

## simpler model with autoregressive effects captured by the ne component
measlesModel2 <- list(
    ne = list(f = ~ 1, weights = W_powerlaw(maxlag=5, from0=TRUE)),
    end = list(f = addSeason2formula(~-1 + ri(), S=1, period=52)),
    family = "NegBin1")
measlesFit2 <- hhh4(measlesWeserEms, measlesModel2)
## omitting the separate AR component simplifies model extensions/selection
## and interpretation of covariate effects (only two predictors left)

plot(measlesFit2, type = "neweights", xlab = "adjacency order")
```

```
## strong decay, again mostly within-district transmission
## (one could also try a purely autoregressive model)
plot(measlesFit2, total = TRUE,
     legend.args = list(legend = c("epidemic", "endemic")))
## almost the same RMSE as with separate AR and NE effects
c(rmse1 = sqrt(mean( residuals(measlesFit, "response")^2 )),
      # = sqrt(mean( scores(measlesFit, "ses")           ))
  rmse2 = sqrt(mean(residuals(measlesFit2, "response")^2)))

}
```

---

hhh4_W_utils *Extract Neighbourhood Weights from a Fitted* hhh4 *Model*

Description

The getNEweights function extracts the (fitted) weight matrix/array from a "hhh4" object, after scaling and normalization. The coefW function extracts the coefficients of parametric neighbourhood weights from a hhh4 fit (or directly from a corresponding coefficient vector), i.e., coefficients whose names begin with "neweights".

Usage

```
getNEweights(object, pars = coefW(object),
             scale = ne$scale, normalize = ne$normalize)
coefW(object)
```

Arguments

object	an object of class "hhh4". coefW also works with the coefficient vector.
pars	coefficients for parametric neighbourhood weights, such as for models using W_powerlaw. Defaults to the corresponding point estimates in object.
scale, normalize	parameters of the ne component of hhh4.

Value

getNEweights returns a square numeric matrix.

coefW returns the "neweights" subset of the coefficient vector with the prefix removed from its names.

Author(s)

Sebastian Meyer

See Also

example(W_powerlaw)

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.

`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).

Usage

```
data("imdepi")
```

Format

`imdepi` is an object of class "`epidataCS`" (a list with components `events`, `stgrid`, `W` and `qmatrix`).

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 https://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 (stateD). It was obtained as the "UnaryUnion" of Germany's districts (districtsD) as at 2009-01-01, simplified by the "modified Visvalingam" algorithm (level 6.6%) available at https://MapShaper.org (v. 0.1.17). The objects districtsD and stateD are contained in system.file("shapes", "districtsD.RData", package="surveillance").

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 at the Department of Hygiene and Microbiology, Julius-Maximilians-Universität Würzburg, Germany (https://www.hygiene.uni-wuerzburg.de/meningococcus/). Thanks to Dr. Johannes Elias and Prof. Dr. Ulrich Vogel for providing the data.

Shapefile of Germany's districts as at 2009-01-01: German Federal Agency for Cartography and Geodesy, Frankfurt am Main, Germany, https://gdz.bkg.bund.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:10.1111/j.15410420.2011.01684.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)

# Summarize event data
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")))

# or manually (no legends, no account for tied locations)
plot(imdepi$W, lwd=2, asp=1)
plot(imdepi$events, pch=c(3,4)[imdepi$events$type], cex=0.8,
     col=c("indianred", "darkblue")[imdepi$events$type], add=TRUE)

## 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)
```

---

imdepifit *Example* twinstim *Fit for the* imdepi *Data*

Description

data("imdepifit") is a twinstim model fitted to the imdepi data.

Usage

```
data("imdepifit")
```

Format

an object of class "twinstim" obtained from the following call using data(imdepi):

```
twinstim(endemic = addSeason2formula(~offset(log(popdensity)) +
    I(start/365 - 3.5), S = 1,
    period = 365, timevar = "start"),
    epidemic = ~type + agegrp,
    siaf = siaf.gaussian(),
    data = imdepi, subset = !is.na(agegrp),
    optim.args = list(control = list(reltol = sqrt(.Machine$double.eps))),
    model = FALSE, cumCIF = FALSE)
```

See Also

common methods for "twinstim" fits, exemplified using imdepifit, e.g., summary.twinstim, plot.twinstim, and simulate.twinstim

Examples

```
data("imdepi", "imdepifit")

## how this fit was obtained
imdepifit$call
```

`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: https://survstat.rki.de/. Queried on 25 July 2007.

Examples

```
data(influMen)
plot(influMen, as.one=FALSE, same.scale=FALSE)
```

`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).

Currently the only supported class is "owin" from package **spatstat.geom**.

Usage

```
intersectPolyCircle(object, center, radius, ...)

## S3 method for class 'owin'
intersectPolyCircle(object, center, radius, npoly = 32, ...)
```

Arguments

object	a polygonal domain of one of the supported classes.
center, radius, npoly	see discpoly.
...	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

```
letterR <- surveillance:::LETTERR  # an "owin" (internally used for checks)
plot(letterR, axes = TRUE)
plot(intersectPolyCircle(letterR, center = c(-1, 2), radius = 2),
     add = TRUE, col = 4, lwd = 3)
```

`isoWeekYear` *Find ISO Week and Year of Date Objects*

Description

The function `isoWeekYear` extracts the year and week of a `Date` according to the ISO 8601 specification.

Usage

```
isoWeekYear(Y, M, D)
```

Arguments

`Y`	year(s) or a Date/POSIXt object. Can be a vector.
`M`	month(s), only used if `Y` is not a Date/POSIXt object.
`D`	day(s), only used if `Y` is not a Date/POSIXt object.

Value

A list with entries `ISOYear` and `ISOWeek` containing the corresponding results.

Note

As from **surveillance** 1.17.0, this function simply calls `strftime` with format strings `"%G"` and `"%V"`, respectively, as this is nowadays (R >= 3.1.0) also supported on Windows.

Examples

```
dates <- as.Date(c("2002-12-31","2003-01-01","2003-01-06"))
isoWeekYear(dates)

## the same using numeric inputs:
isoWeekYear(Y = c(2002, 2003, 2003), M = c(12, 1, 1), D = c(31, 1, 6))
```

knox *Knox Test for Space-Time Interaction*

Description

Given temporal and spatial distances as well as corresponding critical thresholds defining what "close" means, the function knox performs Knox (1963, 1964) test for space-time interaction. The corresponding p-value can be calculated either by the Poisson approximation or by a Monte Carlo permutation approach (Mantel, 1967) with support for parallel computation via plapply. There is a simple plot-method showing a truehist of the simulated null distribution together with the expected and observed values. This implementation of the Knox test is due to Meyer et al. (2016).

Usage

```
knox(dt, ds, eps.t, eps.s, simulate.p.value = TRUE, B = 999, ...)

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

Arguments

dt, ds — numeric vectors containing temporal and spatial distances, respectively. Logical vectors indicating temporal/spatial closeness may also be supplied, in which case eps.t/eps.s is ignored. To test for space-time interaction in a single point pattern of n events, these vectors should be of length $n * (n-1)/2$ and contain the pairwise event distances (e.g., the lower triangle of the distance matrix, such as in "dist" objects). Note that there is no special handling of matrix input, i.e., if dt or ds are matrices, all elements are used (but a warning is given if a symmetric matrix is detected).

eps.t, eps.s — Critical distances defining closeness in time and space, respectively. Distances lower than or equal to the critical distance are considered ""close"".

simulate.p.value — logical indicating if a Monte Carlo permutation test should be performed (as per default). Do not forget to set the .Random.seed via an extra .seed argument if reproducibility is required (see the ... arguments below). If simulate.p.value = FALSE, the Poisson approximation is used (but see the note below).

B — number of permutations for the Monte Carlo approach.

... — arguments configuring plapply: .parallel, .seed, and .verbose. By default, no parallelization is performed (.parallel = 1), and a progress bar is shown (.verbose = TRUE).
For the plot-method, further arguments passed to truehist.

x — an object of class "knox" as returned by the knox test.

Value

an object of class "knox" (inheriting from "htest"), which is a list with the following components:

method	a character string indicating the type of test performed, and whether the Poisson approximation or Monte Carlo simulation was used.
data.name	a character string giving the supplied dt and ds arguments.
statistic	the number of close pairs.
parameter	if simulate.p.value = TRUE, the number B of permutations, otherwise the lambda parameter of the Poisson distribution, i.e., the same as null.value.
p.value	the p-value for the test. In case simulate.p.value = TRUE, the p-value from the Poisson approximation is still attached as an attribute "Poisson".
alternative	the character string "greater" (this is a one-sided test).
null.value	the expected number of close pairs in the absence of space-time interaction.
table	the contingency table of dt <= eps.t and ds <= eps.s.

The plot-method invisibly returns NULL.

A toLatex-method exists, which generates LaTeX code for the contingency table associated with the Knox test.

Note

The Poisson approximation works well if the proportions of close pairs in both time and space are small (Kulldorff and Hjalmars, 1999), otherwise the Monte Carlo permutation approach is recommended.

Author(s)

Sebastian Meyer

References

Knox, G. (1963): Detection of low intensity epidemicity: application to cleft lip and palate. *British Journal of Preventive & Social Medicine*, **17**, 121-127.

Knox, E. G. (1964): The detection of space-time interactions. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, **13**, 25-30.

Kulldorff, M. and Hjalmars, U. (1999): The Knox method and other tests for space-time interaction. *Biometrics*, **55**, 544-552.

Mantel, N. (1967): The detection of disease clustering and a generalized regression approach. *Cancer Research*, **27**, 209-220.

Meyer, S., Warnke, I., Rössler, W. and Held, L. (2016): Model-based testing for space-time interaction using point processes: An application to psychiatric hospital admissions in an urban area. *Spatial and Spatio-temporal Epidemiology*, **17**, 15-25. doi:10.1016/j.sste.2016.03.002. Eprint: https://arxiv.org/abs/1512.09052.

See Also

The function `mantel.randtest` in package **ade4** implements Mantel's (1967) space-time interaction test, i.e., using the Pearson correlation between the spatial and temporal distances of all event pairs as the test statistic, and assessing statistical significance using a Monte Carlo permutation approach as with `simulate.p.value` here in the `knox` function. To combine information from different scales `eps.t` and `eps.s` while also handling edge effects, the space-time K-function test available via `stKtest` can be used. Function `epitest` tests epidemicity in a "`twinstim`" point process model.

Examples

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

## Perform the Knox test using the Poisson approximation
knoxtest <- knox(
    dt = dist(imdepiB$events$time), eps.t = 30,
    ds = dist(coordinates(imdepiB$events)), eps.s = 50,
    simulate.p.value = FALSE
)
knoxtest
## The Poisson approximation works well for these data since
## the proportion of close pairs is rather small (204/56280).

## contingency table in LaTeX
if (requireNamespace("xtable"))
  toLatex(knoxtest)


## Obtain the p-value via a Monte Carlo permutation test
knoxtestMC <- knox(
    dt = dist(imdepiB$events$time), eps.t = 30,
    ds = dist(coordinates(imdepiB$events)), eps.s = 50,
    simulate.p.value = TRUE, B = 49  # limited here for speed
    ## optionally: compute permutations in parallel
    ## , .parallel = 2, .verbose = FALSE
)
knoxtestMC
plot(knoxtestMC)
```

`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 re-uses fragments from the ks.test source file https://svn.R-project.org/R/trunk/src/library/stats/R/ks.test.R, with Copyright (C) 1995-2022 The R Core Team, available under GPL-2 (or later), and C functionality from the source file https://svn.R-project.org/R/trunk/src/library/stats/src/ks.c, partially based on code published in Marsaglia et al. (2003), with Copyright (C) 1999-2022 The R Core Team, also available under GPL-2 (or later).

References

George Marsaglia and Wai Wan Tsang and Jingbo Wang (2003): Evaluating Kolmogorov's distribution. *Journal of Statistical Software*, **8** (18). doi:10.18637/jss.v008.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 Items for* spplot

Description

Generate sp.layout items for use by spplot or plot these items directly in the traditional graphics system. Function layout.labels draws labels at the coordinates of the spatial object, and layout.scalebar returns a labeled scale bar.

Usage

```
layout.labels(obj, labels = TRUE, plot = FALSE)

layout.scalebar(obj, corner = c(0.05, 0.95), scale = 1,
                labels = c(0, scale), height = 0.05,
                pos = 3, ..., plot = FALSE)
```

Arguments

obj — an object inheriting from a "Spatial" class.

labels — specification of the labels. For layout.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.

For layout.scalebar, a character vector of length two giving the labels to be put above the left and right ends of the scale bar.

corner — the location of the scale bar in the unit square, where c(0,0) refers to the bottom left corner. By default, the scale bar is placed in the top left corner (with a small buffer).

scale — the width of the scale bar in the units of proj4string(obj). If identical(FALSE, is.projected(obj)) (i.e., obj has longlat coordinates), scale is interpreted in kilometres.

height — the height of the scale bar, see layout.scale.bar.

pos — a position specifier for the labels (see text). By default, the labels are plotted above the scale bar.

... — further arguments for panel.text (if plot = FALSE) or text (if plot = TRUE) to change the style of the labels, e.g., cex, col, and font.

plot — logical indicating if the layout item should be plotted using the traditional graphics system. By default (FALSE), a list for subsequent use by spplot is returned.

Value

For `layout.labels`, a single `sp.layout` item, which is a list with first element `"panel.text"` and subsequent elements being arguments to that function based on the `labels` specification.

For `layout.scalebar`, a list of `sp.layout` items comprising the polygonal scale bar and the labels.

If these layout functions are called with `plot = TRUE`, the item is plotted directly using traditional graphics functions and `NULL` is returned.

Author(s)

Sebastian Meyer

Examples

```
## districts in the Regierungsbezirk Weser-Ems (longlat coordinates)
data("measlesWeserEms")
mapWE <- measlesWeserEms@map
li1 <- layout.labels(mapWE, labels = list(font=2, labels="GEN"))
li2 <- layout.scalebar(mapWE, corner = c(0.05, 0.05), scale = 20,
                       labels = c("0", "20 km"))
spplot(mapWE, zcol = "AREA", sp.layout = c(list(li1), li2),
       col.regions = rev(heat.colors(100)), scales = list(draw = TRUE))

## districts in Bavaria (projected coordinates)
load(system.file("shapes", "districtsD.RData", package = "surveillance"))
bavaria <- districtsD[substr(row.names(districtsD), 1, 2) == "09", ]
sb <- layout.scalebar(bavaria, corner = c(0.75,0.9), scale = 50,
                      labels = c("0", "50 km"), cex = 0.8)
spplot(bavaria, zcol = "POPULATION", sp.layout = sb,
       xlab = "x [km]", ylab = "y [km]", scales = list(draw = TRUE),
       col.regions = rev(heat.colors(100)))

## these layout functions also work in the traditional graphics system
par(mar = c(0,0,0,0))
plot(bavaria, col = "lavender")
layout.scalebar(bavaria, corner = c(0.75, 0.9), scale = 50,
                labels = c("0", "50 km"), plot = TRUE)
layout.labels(bavaria, labels = list(cex = 0.8,
              labels = substr(bavaria$GEN, 1, 3)), plot = TRUE)
```

`linelist2sts` *Convert Dates of Individual Case Reports into a Time Series of Counts*

Description

The function is used to convert an individual line list of cases to an aggregated time series of counts based on event date information of the cases.

Usage

```
linelist2sts(linelist,dateCol,
              aggregate.by=c("1 day", "1 week", "7 day", "1 week",
"1 month",  "3 month", "1 year"),
              dRange=NULL,
              epochInPeriodStr=switch(aggregate.by, "1 day"="1",
"1 week"="%u", "1 month"="%d","3 month"="%q","1 year"="%j"),
              startYearFormat=switch(aggregate.by,"1 day"="%Y",
"7 day"="%G", "1 week"="%G","1 month"="%Y","3 month"="%Y","1 year"="%Y"),
              startEpochFormat=switch(aggregate.by,"1 day"="%j",
"7 day"="%V", "1 week"="%V", "1 month"="%m", "3 month"="%Q", "1 year"="1")
)
```

Arguments

`linelist` A `data.frame` containing the line list of cases.

`dateCol` A character string stating the column name in `linelist` which contains the event occurrence information (as a vector of `Dates`) 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 date for doing the aggregation. If not specified these dates are extracted automatically by taking `range(D[,dateCol])` and adjust these according to `aggregate.by` (e.g. always first of a month).

`epochInPeriodStr` `strptime` compatible format string to use for determining how a date is placed within the epoch. This is, e.g., used to move the `dRange` epochs to the beginning of the period. Example: In case of weekly aggregation the "%u" determines which day within the week (Monday is day 1) we have. See `strptime` for further details.

`startYearFormat` `strptime` compatible format string to use for determining how the `start` entry of the `sts` object is generated. Usually the provided defaults are sufficient.

`startEpochFormat` `strptime` compatible format string to use for determining how the `start` entry of the `sts` object is generated. Usually the provided defaults are sufficient.

Details

The date range is automatically extended such that the starting and ending dates are always the first epoch within the period, i.e. for aggregation by week it is moved to Mondays. This is controlled by the `epochInPeriodStr` parameter.

Please note that the formatting strings are implemented by the `formatDate` function, which uses `strptime` formatting strings as well as formatting of quarters via "%Q", "%OQ" and "%q".

Value

The function returns an object of class "`sts`". The `freq` slot might not be appropriate.

Author(s)

Michael Höhle

See Also

seq.Date, strptime, formatDate

Examples

```
#Load O104 outbreak data
data("husO104Hosp")

#Convert line list to an sts object
sts <- linelist2sts(husO104Hosp, dateCol="dHosp", aggregate.by="1 day")

#Check that the number of cases is correct
all.equal(sum(observed(sts)),nrow(husO104Hosp))

#Plot the result
plot(sts,xaxis.tickFreq=list("%d"=atChange,"%m"=atChange),
           xaxis.labelFreq=list("%d"=at2ndChange),
           xaxis.labelFormat="%d %b",
           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 is greater than, say, 50 it's necessary to increase the value of `g`.
`outcomeFun`	A hook `function(k,n)` to compute all possible outcome states to compute the likelihood ratio for. If `NULL` then the internal default function `surveillance:::outcomeFunStandard` 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 homogeneous (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): Online change-point detection in categorical time series. In: T. Kneib and G. Tutz (Eds.), Statistical Modelling and Regression Structures - Festschrift in Honour of Ludwig Fahrmeir, Physica-Verlag, pp. 377-397. Preprint available as https://staff.math.su.se/hoehle/pubs/hoehle2010-preprint.pdf

Höhle, M. and Mazick, A. (2010): Aberration detection in R illustrated by Danish mortality monitoring. In: T. Kass-Hout and X. Zhang (Eds.), Biosurveillance: A Health Protection Priority, CRC-Press. Preprint available as https://staff.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):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]))

#######################################################
#Run length of a time varying negative binomial CUSUM
#######################################################

mu0 <- matrix(5*sin(2*pi/52 * 1:104) + 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: https://survstat.rki.de/; m1 and m3 were queried on 10 November 2004. The rest during September 2004.

Examples

```
data(k1)
survResObj <- algo.rki1(k1, control=list(range=27:192))
plot(survResObj, "RKI 1", "k1")
```

---

magic.dim *Compute Suitable k1 x k2 Layout for Plotting*

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

numeric 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**.

makeControl *Generate* control *Settings for an* hhh4 *Model*

Description

Generate control Settings for an hhh4 Model

Usage

```
makeControl(f = list(~1), S = list(0, 0, 1), period = 52, offset = 1, ...)
```

Arguments

f, S, period	arguments for addSeason2formula defining each of the three model formulae in the order (ar, ne, end). Recycled if necessary within mapply.
offset	multiplicative component offsets in the order (ar, ne, end).
...	further elements for the hhh4 control list. The family parameter is set to "NegBin1" by default.

Value

a list for use as the control argument in hhh4.

Examples

```
makeControl()

## a simplistic model for the fluBYBW data
## (first-order transmission only, no district-specific intercepts)
data("fluBYBW")
mycontrol <- makeControl(
    f = list(~1, ~1, ~t), S = c(1, 1, 3),
    offset = list(population(fluBYBW)),  # recycled -> in all components
    ne = list(normalize = TRUE),
    verbose = TRUE)
str(mycontrol)

if (surveillance.options("allExamples"))
## fit this model
fit <- hhh4(fluBYBW, mycontrol)
```

marks *Import from package* **spatstat.geom**

Description

The generic function marks is imported from package **spatstat.geom**. See spatstat.geom::marks for **spatstat.geom**'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). These data are illustrated and analyzed in Meyer et al. (2017, Section 5), see `vignette("hhh4_spacetime")`.

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 (LSN, 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: https://survstat.rki.de/.

A shapefile of Germany's districts as of 01/01/2009 was obtained from the German Federal Agency for Cartography and Geodesy (https://gdz.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 https://MapShaper.org.

Population numbers were obtained from the Federal Statistical Office of Lower Saxony (LSN).

Vaccination coverage was obtained from the public health department of Lower Saxony (NLGA, "Impfreport").

References

Meyer, S., Held, L. and Höhle, M. (2017): Spatio-temporal analysis of epidemic phenomena using the R package **surveillance**. *Journal of Statistical Software*, **77** (11), 1-55. doi:10.18637/jss.v077.i11

Examples

```
## old "disProg" object
data("measles.weser")
measles.weser
plot(measles.weser, as.one=FALSE)

## new "sts" object (with corrections)
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: https://survstat.rki.de/; 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. doi:10.1017/S0950268810001664

See Also

MMRcoverageDE

Examples

```
data(measlesDE)
plot(measlesDE)

## aggregate to bi-weekly intervals
measles2w <- aggregate(measlesDE, nfreq = 26)
plot(measles2w, ~time)

## use a date index for nicer x-axis plotting
epoch(measles2w) <- seq(as.Date("2005-01-03"), by = "2 weeks",
                        length.out = nrow(measles2w))
plot(measles2w, ~time)
```

---

meningo.age *Meningococcal infections in France 1985-1997*

Description

Monthly counts of meningococcal infections in France 1985-1997. Here, the data is split into 4 age groups (<1, 1-5, 5-20, >20).

Usage

```
data(meningo.age)
```

Format

An object of class `disProg` with 156 observations in each of 4 age groups.

week Month index

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 1985-97")
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: the 16 federal states are followed by the total of Germany, as well as the total of West and East Germany.

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

---

momo *Danish 1994-2008 all-cause mortality data for eight age groups*

Description

Weekly number of all cause mortality from 1994-2008 in each of the eight age groups <1, 1-4, 5-14, 15-44, 45-64, 65-74, 75-84 and 85+ years, see Höhle and Mazick (2010).

Usage

```
data(momo)
```

Format

An object of class "sts" containing the weekly number of all-cause deaths in Denmark, 1994-2008 (782 weeks), for each of the eight age groups <1, 1-4, 5-14, 15-44, 45-64, 65-74, 75-84 and 85+ years. A special feature of the EuroMOMO data is that weeks follow 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 eight age groups. These are yearly data obtained from the StatBank Denmark.

Source

European monitoring of excess mortality for public health action (EuroMOMO) project. https://www.euromomo.eu/.

Department of Epidemiology, Statens Serum Institute, Copenhagen, Denmark StatBank Denmark, Statistics Denmark, https://www.statistikbanken.dk/

References

Höhle, M. and Mazick, A. (2010). Aberration detection in R illustrated by Danish mortality monitoring. In T. Kass-Hout and X. Zhang (eds.), *Biosurveillance: A Health Protection Priority*, chapter 12. Chapman & Hall/CRC.
Preprint available at https://staff.math.su.se/hoehle/pubs/hoehle_mazick2009-preprint.pdf

Examples

```
data("momo")
momo

## show the period 2000-2008 with customized x-axis annotation
## (this is Figure 1 in Hoehle and Mazick, 2010)
oopts <- surveillance.options("stsTickFactors" = c("%G" = 1.5, "%Q"=.75))
plot(momo[year(momo) >= 2000,], ylab = "", xlab = "Time (weeks)",
     par.list = list(las = 1), col = c(gray(0.5), NA, NA),
     xaxis.tickFreq = list("%G"=atChange, "%Q"=atChange),
     xaxis.labelFreq = list("%G"=atChange), xaxis.labelFormat = "%G")
surveillance.options(oopts)


if (surveillance.options("allExamples")) {

## stratified monitoring from 2007-W40 using the Farrington algorithm
phase2 <- which(epoch(momo) >= "2007-10-01")
momo2 <- farrington(momo, control = list(range=phase2, alpha=0.01, b=5, w=4))
print(colSums(alarms(momo2)))
plot(momo2, col = c(8, NA, 4), same.scale = FALSE)

## stripchart of alarms (Figure 5 in Hoehle and Mazick, 2010)
plot(momo2, type = alarm ~ time, xlab = "Time (weeks)", main = "",
     alarm.symbol = list(pch=3, col=1, cex=1.5))

}
```

`multiplicity` *Import from package* **spatstat.geom**

Description

The generic function `multiplicity` is imported from package **spatstat.geom**. See `spatstat.geom::multiplicity` for **spatstat.geom**'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.geom** is intended to count the number of duplicates of each element of an object. **spatstat.geom** 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.geom**. 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).

Usage

```
nbOrder(neighbourhood, maxlag = Inf)
```

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 (`Inf`) means no truncation (but orders cannot be larger than the number of regions minus 1), whereas `maxlag = 1` just returns the input neighbourhood matrix (converted to binary integer mode).

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.

Author(s)

Sebastian Meyer

See Also

`nblag` from the **spdep** package

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
nblags <- nbOrder(adjmat)
nblags
```

---

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 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, the 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, m.interpretation=c("hoehle_anderheiden2014", "lawless1994"),
        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,
                      cp_order=c("zero","one"),
                      Wextra=NULL,
                      logLambda=c("iidLogGa","tps","rw1","rw2"),
                      responseDistr=c("poisson", "negbin"),
                      mcmc=c(burnin=2500, sample=10000, thin=1, adapt=1000,
                             store.samples=FALSE)),
            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 for doing the nowcasting. 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).

`"unif"`

`"bayes.notrunc"` A Bayesian procedure ignoring truncation.

`"bayes.notrunc.bnb"` A fast Bayesian procedure ignoring truncation and which calculates the adjustment per-time (i.e. ignoring other delays) using the negative binomial.

`"lawless"` A discretized version of the Gaussian predictive distribution suggested in Lawless (1994).

`"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).

"bayes.trunc.ddcp" Fully Bayesian method allowing for change-points in the delay distribution, e.g., due to speed-ups in the reporting process. A discrete-survival model is used for the delay distribution. Details of the methods are described in HadH (2014). Note: This method requires that the JAGS program is installed on the system.

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 Size of the moving window for the estimation of the delay distribution. Default: NULL, i.e. take all values at all times. Otherwise: a positive integer equal to or greater than D such that only values from a sliding window are used. The shape of the window depends on the value of m.interpretation.

m.interpretation

This parameter controls the interpretation of the sliding window used to estimate the delay distribution. If m.interpretation="hoehle_anderheiden2014" (Default) then the sliding window is defined as a horizontal cut in the reporting triangle, i.e. the values for the delay estimation originate from reports occurring during (now-m):now. This means that the estimation of long delays is based on fewer observations than the estimation of the short delays, hence, the long delay estimates are subject to more variability. If for example $m = D$ then the estimate for a delay of $d = D$ is based on only one observation. The advantage of this choice is that one explicitly knows which time period all observations originate from. For details see Section 3 of Höhle and an der Heiden (2014).

Alternatively, when m.interpretation="lawless1994", the cut in the reporting triangle is made such that each delay d is estimated based on the same number of observations ($m + 1$). This means that in order to estimate the delay for d days, a sliding rectangle of length $m+1$ containing the reports which occurred during (now-m-d):now. See Fig. 2 in Lawless (1994) for details. Note: A warning is given is method="lawless", but m.interpretation is not.

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 parameter 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

`cp_order` Either `"zero"` (Default) or `"one"`. This is the degree of the TPS spline for the baseline hazard, which is formed by the changepoints. Order zero corresponds to the dummy variables of the change-points being simply zero or one. In case a 1st order polynomial is chosen, this allows the delay distribution to change towards faster or slow reporting as time progresses (until the next change-point). The later can be helpful in very dynamic epidemic situations where a lot of cases suddenly appear overwhelming the surveillance system infrastructure.

`Wextra` An additional design matrix part to be joined onto the part originating from the change-points. Altogether, the column bind of these two quantities will be $W_{t,d}$. This allows one to include, e.g., day of the week effects or holidays.

`logLambda` Prior on the spline. One of `c("iidLogGa","tps","rw1","rw2")`.

`respDistr` Response distribution of $n_{t,d}$ in the reporting triangle. Default is `"poisson"`. An experimental alternative is to use `"negbin"`.

`tau.gamma`

`eta.mu` Vector of coefficients describing the mean of the prior normal distribution of the regression effects in the discrete time survival model.

`eta.prec` A precision matrix for the regression effects in the discrete time survival model.

`mcmc` A named vector of length 5 containing burn-in (default: 2500), number of samples (10000), thinning (1) and adaptation (1000) for the three MCMC chains which are ran. The values are passed on to `run.jags`. The fifth argument `store.samples` denotes if the output of the JAGS sampling should be included as part of the returned `stsNC` object. Warning: If `TRUE` (Default: `FALSE`) the size of the returned object might increase substantially.

score Compute scoring rules. Default: `FALSE`. The computed scores are found in the `SR` slot of the result.

predPMF Boolean whether to return 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 is 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 "rjparallel" method of run.jags, which additionally requires the **rjags** package. You need to manually install JAGS on your computer for the package to work – see https://mcmc-jags.sourceforge.io/ 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* 70(4):993-1002. doi:10.1111/biom.12194.
A preprint is available as https://staff.math.su.se/hoehle/pubs/hoehle_anderheiden2014-preprint.pdf.

Günther, F. and Bender, A. and Katz, K. and Küchenhoff, H. and Höhle, M. (2020): Nowcasting the COVID-19 pandemic in Bavaria. *Biometrical Journal*. doi:10.1002/bimj.202000112
Preprint available at doi:10.1101/2020.06.26.20140210.

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))

## Not run:
### Using runjags to do a Bayesian model with changepoint(s)
### -- this might take a while
nc.control.ddcp <- modifyList(nc.control,
                    list(gd.prior.kappa=0.1,
                         ddcp=list(ddChangepoint=as.Date(c("2011-05-23")),
                             logLambda="tps",
                             tau.gamma=1,
                             mcmc=c(burnin=1000,sample=1000,thin=1,
                                    adapt=1000,store.samples=FALSE))))

nc.ddcp <- nowcast(now=t.repTriangle,when=when,
               dEventCol="dHosp",dReportCol="dReport",
               data=husO104Hosp, aggregate.by="1 day",
               method="bayes.trunc.ddcp", D=15,
                   control=nc.control.ddcp)

plot(nc.ddcp,legend.opts=NULL,
     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))

lambda <- attr(delayCDF(nc.ddcp)[["bayes.trunc.ddcp"]],"model")$lambda
showIdx <- seq(which( max(when) == epoch(nc.ddcp))) #seq(ncol(lambda))
matlines( showIdx,t(lambda)[showIdx,],col="gray",lwd=c(1,2,1),lty=c(2,1,2))
legend(x="topright",c(expression(lambda(t)),"95% CI"),col="gray",lwd=c(2,1),lty=c(1,2))

## End(Not run)
```

---

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. range Vector of indices in the observed slot to monitor. theta0 In-control parameters of the paired binary CUSUM. theta1 Out-of-control parameters of the paired binary CUSUM. h1 Primary control limit (=threshold) of 1st CUSUM. h2 Primary control limit (=threshold) of 2nd CUSUM. h11 Secondary limit for 1st CUSUM. h22 Secondary limit for 2nd CUSUM.
p	Vector giving the probability of the four different possible 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 each 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))
if (surveillance.options("allExamples"))
#Compute ARL using slow, non-sparse matrix operations
pairedbinCUSUM.runlength(p,w1=c(-1,37,-9,29),w2=c(-1,7),h1=70,h2=32,
                         h11=38,h22=17)

#Sparse computations can be considerably (!) faster
pairedbinCUSUM.runlength(p,w1=c(-1,37,-9,29),w2=c(-1,7),h1=70,h2=32,
                         h11=38,h22=17,sparse=TRUE)

#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, ylab="CUSUM Statistic")



#######################################################################
#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
opar <- 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)
par(opar)

#####################################################################
# 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 caused 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

The difference between mean scores from model 1 and mean scores from model 2 is used as the test statistic. 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 from models 1 and 2, respectively.
nPermutation	number of Monte Carlo replicates.
plot	logical indicating if a truehist of the nPermutation permutation test statistics should be plotted with a vertical line marking the observed difference of the means. To customize the histogram, plot can also be a list of arguments for truehist replacing internal defaults.
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 1 or 2 with probability 0.5. We then compute the respective difference in mean for model 1 and 2 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 with contributions by 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** (10), 1118-1136. doi:10.1002/sim.4177

See Also

Package **coin** for a comprehensive permutation test framework.

Examples

```
permutationTest(rnorm(50, 1.5), rnorm(50, 1), plot = TRUE)
```

`pit` *Non-Randomized Version of the PIT Histogram (for Count Data)*

Description

See Czado et al. (2009).

Usage

```
pit(x, ...)
## Default S3 method:
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. Due to a historical bug, `relative=TRUE` (the default) actually plots a density histogram while `relative=FALSE` plots relative frequencies.
`...`	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`. Otherwise, no plot will be produced.

Value

an object of class `"pit"`, which inherits from 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. and Held, L. (2009): Predictive model assessment for count data. *Biometrics*, **65** (4), 1254-1261. doi:10.1111/j.15410420.2009.01191.x

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,
        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 = FALSE),
                    pit(x, pdistrs[[2]], plot = FALSE)))
```

---

plapply *Verbose and Parallel* lapply

Description

Verbose and parallelized version of `lapply` wrapping around `mclapply` and `parLapply` in the base package **parallel**. This wrapper can take care of the `.Random.seed` and print progress information (not for cluster-based parallelization). With the default arguments it equals `lapply` enriched by a progress bar.

Usage

```
plapply(X, FUN, ...,
        .parallel = 1, .seed = NULL, .verbose = TRUE)
```

Arguments

X, FUN, ...	see lapply.
.parallel	the number of processes to use in parallel operation, or a "cluster" object (see makeCluster). If a number, mclapply (forking) is used on Unix-alikes, whereas on Windows parLapply is used on a newly created cluster of the specified size, which is stopped when exiting the function. By default (.parallel = 1), the basic lapply is used.
.seed	If set (non-NULL), results involving random number generation become reproducible. If using a cluster (see the .parallel argument), clusterSetRNGStream is called with the specified .seed before running parLapply. Otherwise, set.seed(.seed) is called and the RNGkind is changed to "L'Ecuyer-CMRG" if .parallel > 1 (see the section on random numbers in the documentation of mcparallel in package **parallel**). If .seed is non-NULL, the original .Random.seed will be restored on.exit of the function.
.verbose	if and how progress information should be displayed, i.e., what to do on each exit of FUN. This is unsupported and ignored for cluster-based parallelization and primitive FUNctions. The default (TRUE) will show a txtProgressBar (if .parallel = 1 in an interactive R session) or cat(".") (otherwise). Other choices for the dot are possible by specifying the desired symbol directly as the .verbose argument. Alternatively, .verbose may be any custom call or expression to be executed on.exit of FUN and may thus involve any objects from the local evaluation environment.

Value

a list of the results of calling FUN on each value of X.

Author(s)

Sebastian Meyer

See Also

mclapply and parLapply

Examples

```
## example inspired by help("lapply")
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))

## if neither parallel nor verbose then this simply equals lapply()
plapply(x, quantile, probs = 1:3/4, .verbose = FALSE)

## verbose lapply() -- not really useful for such fast computations
res <- plapply(x, quantile, probs = 1:3/4, .verbose = TRUE)
res <- plapply(x, quantile, probs = 1:3/4, .verbose = "|")
res <- plapply(x, quantile, probs = 1:3/4,
               .verbose = quote(cat("length(x) =", length(x), "\n")))
```

```
## setting the seed for reproducibility of results involving the RNG
samp <- plapply(as.list(1:3), runif, .seed = 1)

## parallel lapply()
res <- plapply(x, quantile, probs = 1:3/4, .parallel = 2, .verbose = FALSE)

## using a predefined cluster
library("parallel")
cl <- makeCluster(getOption("cl.cores", 2))
res <- plapply(x, quantile, probs = 1:3/4, .parallel = cl)
stopCluster(cl)
```

`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", zero.policy=zero.policy)`.

Usage

```
poly2adjmat(SpP, ..., zero.policy = TRUE)
```

Arguments

`SpP`	an object inheriting from `"SpatialPolygons"`.
`...`	arguments passed to `poly2nb`. Its `snap` argument might be particularly useful to handle maps with sliver polygons.
`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)]))

    ## a visual check of the district-specific number of neighbours
    plot(fluBYBW@map)
    text(coordinates(fluBYBW@map), labels=rowSums(adjmat==1), font=2, col=2)
}
```

---

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 = "sf", ...)
```

Arguments

SpP	an object of class "SpatialPolygons".
snap	tolerance used to consider coordinates as identical.
method	method to use for unionSpatialPolygons. Defaults to **sf**, 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"))

## Determine districts at the border and check the result on the map
if (requireNamespace("sf")) {
    atBorder <- polyAtBorder(districtsD, method = "sf")
    if (interactive()) plot(districtsD, col = atBorder)
    table(atBorder)
}

## For method = "polyclip", a higher snapping tolerance is required
## to obtain the correct result
if (requireNamespace("polyclip")) {
    atBorder <- polyAtBorder(districtsD, snap = 1e-6, method = "polyclip")
    if (interactive()) plot(districtsD, col = atBorder)
    table(atBorder)
}
```

`primeFactors` *Prime Number Factorization*

Description

Computes the prime number factorization of an integer.

Usage

```
primeFactors(x)
```

Arguments

x	an integer

Value

vector with prime number factorization of x

R0 *Computes reproduction numbers from fitted models*

Description

The S3 generic function R0 defined in package **surveillance** is intended to compute reproduction numbers from fitted epidemic models. The package currently defines a method for the "twinstim" class, which computes expected 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, ...)

simpleR0(object, eta = coef(object)[["e.(Intercept)"]],
         eps.s = NULL, eps.t = NULL, newcoef = NULL)
```

Arguments

object	A fitted epidemic model object for which an R0 method exists.
newevents	an optional data.frame of events for which the 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$R0; however, if trimmed = FALSE, the model environment is required, i.e. object must have been fitted with model = TRUE. For the twinstim method, newevents must at least contain the following columns: the event time (only for trimmed = TRUE) and type (only for multi-type epidemics), the maximum interaction ranges eps.t and eps.s, as well as columns for the marks and stgrid variables used in the epidemic component of the fitted "twinstim" object as stored in formula(object)$epidemic. 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.
eta	a value for the epidemic linear predictor, see details.
eps.s, eps.t	the spatial/temporal radius of interaction. If NULL (the default), the original value from the data is used if this is unique and an error is thrown otherwise.

Details

For the "twinstim" class, the individual-specific expected 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):

$$\mu_j = e^{\eta_j} \cdot \int_{b(\mathbf{0},\delta)} f(\boldsymbol{s}) d\boldsymbol{s} \cdot \int_0^{\epsilon} g(t) dt.$$

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. For a type-specific twinstim, there is an additional factor for the number of event types which can be infected by the type of event j and the interaction functions may be type-specific as well.

Alternatively to the equation above, the trimmed (observed) 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 $(0; \min(T - t_j, \epsilon)]$).

The function simpleR0 computes

$$\exp(\eta) \cdot \int_{b(\mathbf{0},\delta)} f(\boldsymbol{s}) d\boldsymbol{s} \cdot \int_0^{\epsilon} g(t) dt,$$

where η defaults to γ_0 disregarding any epidemic effects of types and marks. It is thus only suitable for simple epidemic twinstim models with epidemic = ~1, a diagonal (or secondary diagonal) qmatrix, and type-invariant interaction functions. simpleR0 mainly exists for use by epitest.

(Numerical) Integration is performed exactly as during the fitting of object, for instance object$control.siaf is queried if necessary.

Value

For the R0 methods, a numeric vector of estimated 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.

For simpleR0, a single number (see details).

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:10.1111/j.15410420.2011.01684.x

Examples

```
## load the 'imdepi' data and a model fit
data("imdepi", "imdepifit")

## calculate individual and type-specific reproduction numbers
R0s <- R0(imdepifit)
tapply(R0s, imdepi$events@data[names(R0s), "type"], summary)

## untrimmed R0 for specific event settings
refevent <- data.frame(agegrp = "[0,3)", type = "B", eps.s = Inf, eps.t = 30)
setting2 <- data.frame(agegrp = "[3,19)", type = "C", eps.s = Inf, eps.t = 14)
newevents <- rbind("ref" = refevent, "event2" = setting2)
(R0_examples <- R0(imdepifit, newevents = newevents, trimmed = FALSE))
stopifnot(all.equal(R0_examples[["ref"]],
                    simpleR0(imdepifit)))


### compute a Monte Carlo confidence interval

## use a simpler model with constant 'siaf' for speed
simplefit <- update(imdepifit, epidemic=~type, siaf=NULL, subset=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))


### R0 for a simple epidemic model
```

```
### without epidemic covariates, i.e., all individuals are equally infectious

mepi1 <- update(simplefit, epidemic = ~1, subset = type == "B",
                model = TRUE, verbose = FALSE)
## using the default spatial and temporal ranges of interaction
(R0B <- simpleR0(mepi1))  # eps.s=200, eps.t=30
stopifnot(identical(R0B, R0(mepi1, trimmed = FALSE)[[1]]))
## assuming smaller interaction ranges (but same infection intensity)
simpleR0(mepi1, eps.s = 50, eps.t = 15)
```

`ranef` *Import from package* **nlme**

Description

The generic functions `ranef` and `fixef` are imported from package **nlme**. See `nlme::ranef` for **nlme**'s own description, and `ranef.hhh4` or `fixef.hhh4` for the added methods for `"hhh4"` models.

`refvalIdxByDate` *Compute indices of reference value using Date class*

Description

The reference values are formed based on computations 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

residualsCT *Extract Cox-Snell-like Residuals of a Fitted Point Process*

Description

Extract the "residual process" (cf. Ogata, 1988) of a fitted point process model specified through the conditional intensity function, for instance a model of class "twinSIR" or "twinstim" (and also "simEpidataCS"). 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, ...)
## S3 method for class 'simEpidataCS'
residuals(object, ...)
```

Arguments

object	an object of one of the aforementioned model classes.
...	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 (and they always are for "simEpidataCS"). 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 calculated 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.

`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 were queried on 19 Feb 2014 from the Survstat@RKI database of the German Robert Koch Institute (https://survstat.rki.de/).

`salmAllOnset` *Salmonella cases in Germany 2001-2014 by data of symptoms onset*

Description

A dataset containing the reported number of cases of Salmonella in Germany 2001-2014 aggregated by data of disease onset. The slot `control` contains a matrix `reportingTriangle$n` with the reporting triangle as described in Salmon et al. (2015).

Usage

```
data(salmAllOnset)
```

Format

A sts-object

References

Salmon, M., Schumacher, D., Stark, K., Höhle, M. (2015): Bayesian outbreak detection in the presence of reporting delays. Biometrical Journal, 57 (6), 1051-1067.

`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` and `multinomialTS` is `TRUE`.

Usage

```
data(salmHospitalized)
```

Format

An "`sts`" object.

Source

The data are queried from the Survstat@RKI database of the German Robert Koch Institute (https://survstat.rki.de/).

`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 were queried from the SurvStat@RKI database of the German Robert Koch Institute (https://survstat.rki.de/). 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.

---

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.

---

scores *Proper Scoring Rules for Poisson or Negative Binomial Predictions*

Description

Proper scoring rules for Poisson or negative binomial predictions of count data are described in Czado et al. (2009). The following scores are implemented: logarithmic score (`logs`), ranked probability score (`rps`), Dawid-Sebastiani score (`dss`), squared error score (`ses`).

Usage

```
scores(x, ...)

## Default S3 method:
scores(x, mu, size = NULL,
       which = c("logs", "rps", "dss", "ses"),
       sign = FALSE, ...)

logs(x, mu, size = NULL)
rps(x, mu, size = NULL, k = 40, tolerance = sqrt(.Machine$double.eps))
dss(x, mu, size = NULL)
ses(x, mu, size = NULL)
```

Arguments

`x`	the observed counts. All functions are vectorized and also accept matrices or arrays. Dimensions are preserved.
`mu`	the means of the predictive distributions for the observations `x`.
`size`	either `NULL` (default), indicating Poisson predictions with mean `mu`, or dispersion parameters of negative binomial forecasts for the observations `x`, parametrized as in `dnbinom` with variance `mu*(1+mu/size)`.
`which`	a character vector specifying which scoring rules to apply. By default, all four proper scores are calculated. The normalized squared error score (`"nses"`) is also available but it is improper and hence not computed by default.
`sign`	a 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.
`...`	unused (argument of the generic).
`k`	scalar argument controlling the finite sum approximation for the `rps` with truncation at `max(x, ceiling(mu + k*sd))`.
`tolerance`	absolute tolerance for the finite sum approximation employed in the `rps` calculation. A warning is produced if the approximation with `k` summands is insufficient for the specified `tolerance`. In this case, increase `k` for higher precision (or use a larger tolerance).

Value

The scoring functions return the individual scores for the predictions of the observations in `x` (maintaining their dimension attributes).

The default `scores`-method applies the selected (`which`) scoring functions (and calculates `sign(x-mu)`) and returns the results in an array (via `simplify2array`), where the last dimension corresponds to the different scores.

Author(s)

Sebastian Meyer and Michaela Paul

References

Czado, C., Gneiting, T. and Held, L. (2009): Predictive model assessment for count data. *Biometrics*, **65** (4), 1254-1261. doi:10.1111/j.15410420.2009.01191.x

See Also

The R package **scoringRules** implements the logarithmic score and the (continuous) ranked probability score for many distributions.

Examples

```
mu <- c(0.1, 1, 3, 6, 3*pi, 100)
size <- 0.5
set.seed(1)
y <- rnbinom(length(mu), mu = mu, size = size)
scores(y, mu = mu, size = size)
scores(y, mu = mu, size = 1)  # ses ignores the variance
scores(y, mu = 1, size = size)

## apply a specific scoring rule
scores(y, mu = mu, size = size, which = "rps")
rps(y, mu = mu, size = size)

## rps() gives NA (with a warning) if the NegBin is too wide
rps(1e5, mu = 1e5, size = 1e-5)
```

`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: https://survstat.rki.de/; Queried on September 2006.

Robert Koch Institut, Epidemiologisches Bulletin 31/2006.

Examples

```
data(shadar)
plot(shadar)
```

sim.pointSource *Simulate Point-Source Epidemics*

Description

Simulation of epidemics which were introduced by point sources. The basis of this programme is a combination of a Hidden Markov Model (to get random timepoints for outbreaks) and a simple model (compare `sim.seasonalNoise`) to simulate the baseline.

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 outbreak at time i if there was one at time i-1, default 0.99.
`r`	probability to get no new outbreak 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 weight for an outbreak which influences the distribution parameter mu, default = 0.

Value

a `disProg` (disease progress) object 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

```
set.seed(123)
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(disProgObj)

## with predefined state chain
state <- rep(c(0,0,0,0,0,0,0,0,1,1), 20)
disProgObj <- sim.pointSource(state = state, K = 1.2)
plot(disProgObj)

## simulate epidemic, send to RKI 1 system, plot, and compute quality values
testSim <- function (..., K = 0, range = 200:400) {
  disProgObj <- sim.pointSource(..., K = K)
  survResults <- algo.call(disProgObj,
    control = list(list(funcName = "rki1", range = range)))
  plot(survResults[[1]], "RKI 1", "Simulation")
  algo.compare(survResults)
}
testSim(K = 2)
testSim(r = 0.5, K = 5)  # larger and more frequent outbreaks
```

`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 weight for an outbreak which influences the distribution parameter mu, default = 0.

Value

an 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")
```

---

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 (requireNamespace("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)
}
```

---

stK *Diggle et al (1995) K-function test for space-time clustering*

Description

The function `stKtest` wraps functions in package **splancs** to perform the K-function based Monte Carlo permutation test for space-time clustering (Diggle et al, 1995) for "`epidataCS`". The implementation is due to Meyer et al. (2016).

Usage

```
stKtest(object, eps.s = NULL, eps.t = NULL, B = 199,
        cores = 1, seed = NULL, poly = object$W)

## S3 method for class 'stKtest'
plot(x, which = c("D", "R", "MC"),
     args.D = list(), args.D0 = args.D, args.R = list(), args.MC = list(),
     mfrow = sort(n2mfrow(length(which))), ...)
```

Arguments

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

eps.s, eps.t	numeric vectors defining the spatial and temporal grids of critical distances over which to evaluate the test. The default (NULL) uses equidistant values from 0 to the smallest eps.s/eps.t value in object$events, but not larger than half the observed spatial/temporal domain.
B	the number of permutations.
cores	the number of parallel processes over which to distribute the requested number of permutations.
seed	argument for set.seed to initialize the random number generator such that results become reproducible (also if cores > 1, see plapply).
poly	the polygonal observation region of the events (as an object handled by xylist). The default object$W might not work since package **splancs** does not support multi-polygons. In this case, the poly argument can be used to specify a substitute.
x	an "stKtest".
which	a character vector indicating which diagnostic plots to produce. The full set is c("D", "D0", "R", "MC"). The special value which = "stdiagn" means to call the associated **splancs** function stdiagn.
args.D, args.D0, args.R, args.MC	argument lists for the plot functions persp (for "D" and "D0"), plot.default ("R"), and truehist ("MC"), respectively, to modify the default settings. Ignored if which = "stdiagn".
mfrow	par-setting to layout the plots. Ignored for which = "stdiagn" and if set to NULL.
...	ignored (argument of the generic).

Value

an object of class "stKtest" (inheriting from "htest"), which is a list with the following components:

method	a character string indicating the type of test performed.
data.name	a character string naming the supplied object.
statistic	the sum U of the standardized residuals $R(s,t)$.
parameter	the number B of permutations.
p.value	the p-value for the test.
pts	the coordinate matrix of the event locations (for stdiagn.
stK	the estimated K-function as returned by stkhat.
seD	the standard error of the estimated $D(s,t)$ as returned by stsecal.
mctest	the observed and permutation values of the test statistic as returned by stmctest.

The plot-method invisibly returns NULL.

Author(s)

Sebastian Meyer

References

Diggle, P. J.; Chetwynd, A. G.; Häggkvist, R. and Morris, S. E. (1995): Second-order analysis of space-time clustering *Statistical Methods in Medical Research*, **4**, 124-136.

Meyer, S., Warnke, I., Rössler, W. and Held, L. (2016): Model-based testing for space-time interaction using point processes: An application to psychiatric hospital admissions in an urban area. *Spatial and Spatio-temporal Epidemiology*, **17**, 15-25. doi:10.1016/j.sste.2016.03.002. Eprint: https://arxiv.org/abs/1512.09052.

See Also

the simple knox test and function epitest for testing "twinstim" models.

Examples

```
if (requireNamespace("splancs")) {
    data("imdepi")
    imdepiB <- subset(imdepi, type == "B")
    mainpoly <- coordinates(imdepiB$W@polygons[[1]]@Polygons[[5]])
    SGRID <- c(10, 25, 50, 100, 150)
    TGRID <- c(1, 7, 14, 21)
    B <- 19  # limited here for speed

    imdBstKtest <- stKtest(imdepiB, eps.s = SGRID, eps.t = TGRID, B = B,
                           poly = list(mainpoly))
    print(imdBstKtest)
    plot(imdBstKtest)
}
```

---

sts-class *Class "sts" – surveillance time series*

Description

This is a lightweight S4 class to implement (multivariate) time series of counts, typically from public health surveillance. The "sts" class supersedes the informal "disProg" class used in early versions of package **surveillance**. Converters are available, see disProg2sts.

The constructor function sts can be used to setup an "sts" object. For areal time series, it can also capture a map of the regions, where the counts originate from. See Section "Slots" below for a description of all class components, and Section "Methods" for a list of extraction, conversion and visualization methods.

Usage

```
sts(observed, start = c(2000, 1), frequency = 52,
    epoch = NULL, population = NULL, map = NULL, ...)
```

Arguments

`observed` a vector (for a single time series) or matrix (one time series per column) of counts. A purely numeric data frame will also do (transformed via `as.matrix`). This argument sets the `observed` slot, which is the core element of the resulting `"sts"` object. It determines the dimensions and colnames for several other slots. The columns ("units") typically correspond to different regions, diseases, or age groups.

`start, frequency` basic characteristics of the time series data just like for simple `"ts"` objects. The (historical) default values correspond to weekly data starting in the first week of 2000. The `epoch` and `epochInYear` methods use the ISO 8601 specification when converting between week numbers and dates, see `isoWeekYear`.

`epoch` observation times, either as an integer sequence (default) or as a `Date` vector (in which case `epochAsDate` is automatically set to `TRUE`).

`population` the population sizes (or fractions) underlying the `observed` counts, especially relevant for multivariate time series (for incidence maps or as offsets in epidemic models). If assumed constant over time, `population` can be supplied as a vector of length the number of columns (regions, groups, etc.) in `observed`. Otherwise, `population` needs to be a matrix of the same `dim`ension as `observed`. The `population` argument is an alias for the corresponding slot `populationFrac`. The default `NULL` value sets equal population fractions across all units.

`map` optional spatial data representing the regions, either of class `"SpatialPolygons"` (or `"SpatialPolygonsDataFrame"`) or of class `"sf"` (requires package **sf**). Its `row.names()` must contain the region IDs to be matched against `colnames(observed)`.

`...` further named arguments with names corresponding to slot names (see the list below). For instance, in the public health surveillance context, the `state` slot is used to indicate outbreaks (default: `FALSE` for all observations). For areal time series data, the `map` and `neighbourhood` slots are used to store the spatial structure of the observation region.

Slots

`epoch`**:** a numeric vector specifying the time of observation, typically a week index. Depending on the `freq` slot, it could also index days or months. Furthermore, if `epochAsDate=TRUE` then `epoch` is the integer representation of `Date`s giving the exact date of the observation.

`freq`**:** number of observations per year, e.g., 52 for weekly data, 12 for monthly data.

`start`**:** vector of length two denoting the year and the sample number (week, month, etc.) of the first observation.

`observed`**:** 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 dimensions as `observed` containing Booleans whether at the specific time point there was an outbreak in the region.

`alarm`**:** matrix with the same dimensions as `observed` specifying whether an outbreak detection algorithm declared a specific time point in the region as having an alarm.

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 (see the Examples for how to infer the latter from the map).

populationFrac: matrix of population fractions or absolute numbers (see multinomialTS below) with dimensions dim(observed).

map: object of class "SpatialPolygons" (or "SpatialPolygonsDataFrame") providing a shape of the areas which are monitored or modelled.

control: list of settings; this is a rather free data type to be returned by the surveillance algorithms.

epochAsDate: a Boolean indicating if the epoch slot corresponds to Dates.

multinomialTS: a Boolean stating whether to interpret the object as observed out of population, i.e. a multinomial interpretation instead of a count interpretation.

Methods

Extraction of slots: There is an extraction (and replacement) method for almost every slot. The name of the method corresponds to the slot name, with three exceptions: the freq slot can be extracted by frequency(), the populationFrac slot is accessed by population(), and the alarm slot is accessed by alarms().

epoch signature(x = "sts"): extract the epoch slot. If the sts object is indexed by dates (epochAsDate = TRUE), the returned vector is of class Date, otherwise numeric (usually the integer sequence 1:nrow(x)).
By explicitly requesting epoch(x, as.Date = TRUE), dates can also be extracted if the sts object is not internally indexed by dates but has a standard frequency of 12 (monthly) or 52 (weekly). The transformation is based on start and freq and will return the first day of each month (freq=12) and the Monday of each week (freq=52), respectively.

frequency signature(x = "sts"): extract the freq slot.

start signature(x = "sts"): extract the start slot.

observed signature(x = "sts"): extract the observed slot.

alarms signature(x = "sts"): extract the alarm slot.

upperbound signature(x = "sts"): extract the upperbound slot.

neighbourhood signature(x = "sts"): extract the neighbourhood slot.

population signature(x = "sts"): extract the populationFrac slot.

control signature(x = "sts"): extract the control slot.

multinomialTS signature(x = "sts"): extract the multinomialTS slot.

Other extraction methods:

dim signature(x = "sts"): extract matrix dimensions of observed. This method also enables nrow(x) and ncol(x).

dimnames signature(x = "sts"): extract the dimnames of the observed matrix. This method also enables rownames(x) and colnames(x).

year signature(x = "sts"): extract the corresponding year of each observation.

epochInYear signature(x = "sts"): extract the epoch number within the year. This corresponds to cycle(as.ts(x)).

[signature(x = "sts"): subset rows (time points) and/or columns (units), see help("[,sts-method").

Transformation methods:

aggregate signature(x = "sts"): see aggregate.sts.

as.data.frame signature(x = "sts"): see as.data.frame.sts.

coerce signature(from="sts", to="ts") and signature(from="ts", to="sts"), to be called via as(stsObj, "ts") (or as.ts(stsObj)) and as(tsObj, "sts"), respectively.

as.xts convert to the **xts** package format, see as.xts.sts.

Visualization methods:

plot signature(x = "sts", y = "missing"): entry point to a collection of plot variants. The type of plot is specified using a formula, see plot.sts for details.

autoplot a **ggplot2** variant of the standard time-series-type plot, see autoplot.sts.

animate see animate.sts.

toLatex see toLatex.sts.

Author(s)

Michael Höhle and Sebastian Meyer

Examples

```
showClass("sts")

## create an sts object from time-series data
salmonellaDF <- read.table(system.file("extdata/salmonella.agona.txt",
                                       package = "surveillance"), header = TRUE)
str(salmonellaDF)
salmonella <- with(salmonellaDF,
                   sts(observed = observed, state = state,
                       start = c(1990, 1), frequency = 52))
salmonella
plot(salmonella)

## these data are also available as a legacy "disProg" object in the package
data(salmonella.agona)
stopifnot(all.equal(salmonella, disProg2sts(salmonella.agona)))


## A typical dataset with weekly counts of measles from several districts
data("measlesWeserEms")
measlesWeserEms

## reconstruct data("measlesWeserEms") from its components
counts <- observed(measlesWeserEms)
map <- measlesWeserEms@map
populationFrac <- population(measlesWeserEms)
weserems_nbOrder <- neighbourhood(measlesWeserEms)
## orders of adjacency can also be determined from the map
if (requireNamespace("spdep")) {
```

```
    stopifnot(identical(weserems_nbOrder,
                        nbOrder(poly2adjmat(map))))
}
mymeasles <- sts(counts, start = c(2001, 1), frequency = 52,
                 population = populationFrac,
                 neighbourhood = weserems_nbOrder, map = map)
stopifnot(identical(mymeasles, measlesWeserEms))
```

---

sts_animate *Animated Maps and Time Series of Disease Counts or Incidence*

Description

The `animate`-method for `sts` objects iterates over time points, plotting maps of the current|cumulative counts|incidence via `stsplot_space`, optionally including a 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.

Usage

```
## S3 method for class 'sts'
animate(object, tps = NULL, cumulative = FALSE,
        population = NULL, at = 10, ...,
        timeplot = list(pos = 1, size = 0.3, fill = TRUE),
        sleep = 0.5, verbose = interactive(), draw = TRUE)
```

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/incidence over time should be plotted. The cumulative incidence is relative to the population from the first time point `tps[1]` throughout the whole animation, while `cumulative=FALSE` computes the incidence from the current population numbers.

`population, at, ...` arguments for `stsplot_space`.

`timeplot` if a list and package **gridExtra** is available, a time series chart of the counts along the selected time points `tps` will be plotted next to the map. The list elements determine both the positioning of this plot (`pos`, `size`, and `fill`) and its appearance. The default `pos=1` and `size=0.3` arguments put the time series plot below the map, using 30% of the total plot height. The logical value `fill` indicates whether to make the panel as big as possible (default: TRUE). An alternative to `fill=FALSE` is to manually specify an `aspect` (ratio) value

in timeplot. Other list elements are arguments for the internal (and currently undocumented) function stsplot_timeSimple. For example, 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), and the boolean as.Date determines whether dates should be put on the x-axis (instead of the tps indexes).

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.

draw — logical indicating if the produced plots at each time point should be drawn directly (the default) or not. The setting draw = FALSE is useful if one would like to manually arrange the plots, which are always returned invisibly in a list of length length(tps).

Value

(invisibly) a list of the length(tps) sequential plot objects. These are of class "gtable" (from **gtable**) if the timeplot is included, otherwise of class "\code{trellis".

Author(s)

Sebastian Meyer

See Also

the other plot types documented in stsplot for static time series plots and maps.

Examples

```
data("measlesWeserEms")

## animate the weekly counts of measles (during weeks 12-16 only, for speed)
if (interactive() && require("animation")) {
    oldwd <- setwd(tempdir())  # to not clutter up the current working dir
    saveHTML(animate(measlesWeserEms, tps=12:16),
             title="Evolution of the measles epidemic in the Weser-Ems region",
             ani.width=500, ani.height=600)
    setwd(oldwd)
}

## Not run:
## animate the weekly incidence of measles (per 100'000 inhabitants),
## and label the time series plot with dates in a specified format
animate(measlesWeserEms, tps=12:16,
        population = measlesWeserEms@map$POPULATION / 100000,
        timeplot = list(as.Date = TRUE,
                        scales = list(x = list(format = "%G/%V"))))

## End(Not run)
```

sts_conversion *Convert an* "sts" *Object to* "ts" *or* "xts"

Description

These methods convert **surveillance** time series objects of class "sts" to the standard "ts" and "xts" classes. Conversion to and from ts is also possible using S4-type coerce methods, e.g., as(x, "ts").

Usage

```
## S3 method for class 'sts'
as.ts(x, ...)

## S3 method for class 'sts'
as.xts(x, order.by = epoch(x, as.Date = TRUE), ...)
```

Arguments

x	an object of class "sts".
order.by	optional Dates indexing the time series. The default is only used for weekly (freq=52) and daily (freq=365) data. If not specified, other frequencies are handled via as.xts.ts.
...	further arguments are ignored by the as.ts method and passed to xts by the as.xts method.

Value

an object of class "ts" or "xts", respectively.

Examples

```
## convert "ts"/"mts" object to "sts"
counts <- matrix(rpois(28*3,10), 28, 3,
                 dimnames = list(NULL, paste("Region", 1:3)))
z <- ts(counts, start = c(1983, 12), frequency = 12)
head(z)
x <- as(z, "sts")
x

## direct "sts" construction gives the same:
(x2 <- sts(counts, start = start(z), frequency = frequency(z)))


## conversion of "sts" objects to the "xts" class
if (requireNamespace("xts")) {
    y <- xts::as.xts(x)
    print(head(y))
```

```
    plot(y)
}
```

---

sts_creation *Simulate Count Time Series with Outbreaks*

Description

Function for simulating a time series and creating an sts object. As the counts are generated using a negative binomial distribution one also gets the (1-alpha) quantile for each timepoint (can be interpreted as an in-control upperbound for in-control values). The baseline and outbreaks are created as in Noufaily et al. (2012).

Usage

```
sts_creation(theta, beta, gamma1, gamma2, m, overdispersion, dates,
  sizesOutbreak, datesOutbreak, delayMax, alpha, densityDelay)
```

Arguments

theta	baseline frequency of reports
beta	time trend
gamma1	seasonality
gamma2	seasonality
m	seasonality
overdispersion	size parameter of rnbinom for the parameterization with mean and dispersion
dates	dates of the time series
sizesOutbreak	sizes of all the outbreaks (vector)
datesOutbreak	dates of all the outbreaks (vector)
delayMax	maximal delay in time units
alpha	alpha for getting the (1-alpha) quantile of the negative binomial distribution at each timepoint
densityDelay	density distribution for the delay

Value

a univariate "sts" time series along the (weekly) dates.

References

Noufaily, A., Enki, D.G., Farrington, C.P., Garthwaite, P., Andrews, N.J., Charlett, A. (2012): An improved algorithm for outbreak detection in multiple surveillance systems. Statistics in Medicine, 32 (7), 1206-1222.

Examples

```
set.seed(12345)
# Time series parameters
scenario4 <- c(1.6,0,0.4,0.5,2)
theta <- 1.6
beta <- 0
gamma1 <-0.4
gamma2 <- 0.5
overdispersion <- 1
m <- 1
# Dates
firstDate <- "2006-01-01"
lengthT=350
dates <- as.Date(firstDate) + 7 * 0:(lengthT - 1)
# Maximal delay in weeks
D=10
# Dates and sizes of the outbreaks
datesOutbreak <- as.Date(c("2008-03-30","2011-09-25"))
sizesOutbreak <- c(2,5)
# Delay distribution
data("salmAllOnset")
in2011 <- which(isoWeekYear(epoch(salmAllOnset))$ISOYear == 2011)
rT2011 <- salmAllOnset@control$reportingTriangle$n[in2011,]
densityDelay <- apply(rT2011,2,sum, na.rm=TRUE)/sum(rT2011, na.rm=TRUE)
# alpha for the upperbound
alpha <- 0.05
# Create the sts with the full time series
stsSim <- sts_creation(theta=theta,beta=beta,gamma1=gamma1,gamma2=gamma2,m=m,
                       overdispersion=overdispersion,
                       dates=dates,
                       sizesOutbreak=sizesOutbreak,datesOutbreak=datesOutbreak,
                       delayMax=D,densityDelay=densityDelay,
                       alpha=alpha)
plot(stsSim)
```

`sts_ggplot` *Time-Series Plots for* "sts" *Objects Using* **ggplot2**

Description

A simple **ggplot2** variant of `stsplot_time`, based on a "tidy" version of the "sts" object via `tidy.sts`. It uses a date axis and thus only works for time series indexed by dates or with a standard frequency (daily, (bi-)weekly, or monthly).

Usage

```
autoplot.sts(object, population = FALSE, units = NULL,
             as.one = FALSE, scales = "fixed", width = NULL, ...)
```

Arguments

object	an object of class "sts".
population	logical indicating whether observed(object) should be divided by population(object). The population argument can also be a scalar, which is used to scale the denominator population(object), i.e., observed(object) is divided by population(object) / population. For instance, if population(object) contains raw population numbers, population = 1000 could be used to plot the incidence per 1000 inhabitants.
units	optional integer or character vector to select the units (=columns of object) to plot. The default (NULL) is to plot all time series.
as.one	logical indicating if all time series should be plotted in one panel with geom_line. By default, the time series are plotted in separate panels (using geom_col).
scales	passed to facet_wrap (for as.one=FALSE). By default, all panels use a common ylim (and xlim).
width	bar width, passed to geom_col. Defaults to 7 for weekly time series.
...	unused (argument of the generic).

Value

a "ggplot" object.

Author(s)

Sebastian Meyer

See Also

stsplot_time for the traditional plots.

Examples

```
data("measlesDE")
plot(measlesDE, units = 1:2)  # traditional graphics

## compare this with a ggplot2-based autoplot()
if (requireNamespace("ggplot2", quietly = TRUE))
    autoplot.sts(measlesDE, units = 1:2)

## weekly incidence: population(measlesDE) gives population fractions,
## which we need to multiply by the total population
if (requireNamespace("ggplot2", quietly = TRUE))
    autoplot.sts(measlesDE, population = 1000000/82314906) +
        ggplot2::ylab("Weekly incidence [per 1'000'000 inhabitants]")
```

sts_observation — *Create an* sts *object with a given observation date*

Description

Function for creating an sts object with a given observation date.

Usage

```
sts_observation(sts, dateObservation, cut = TRUE)
```

Arguments

sts	sts-object we want to set at a previous state. Needs to include a reporting triangle.
dateObservation	Date for which we want the state. Needs to be in the reporting triangle dates.
cut	Boolean indicating whether to have 0 counts after the observation date or to simply cut the sts-object

Value

the modified sts object.

Examples

```
data("salmAllOnset")
salmAllOnsety2014m01d20 <- sts_observation(salmAllOnset,
  dateObservation="2014-01-20",cut=FALSE)
plot(salmAllOnset)
lines(observed(salmAllOnsety2014m01d20),type="h",col="red")
```

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".

coerce 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".

coerce signature(from = "sts", to = "stsNC"): convert an object of class sts to class stsNC.

reportingTriangle signature(x = "stsNC"): extract the reportingTriangle slot of an stsNC object.

delayCDF signature(x = "stsNC"): extract the delayCDF slot of an stsNC object.

score signature(x = "stsNC"): extract the scoring rules result slot of an stsNC object.

predint signature(x = "stsNC"): extract the prediction interval slot of an stsNC object.

Author(s)

M. Höhle

stsNClist_animate *Animate a Sequence of Nowcasts*

Description

Animate a sequence of nowcasts stored as a list.

Usage

```
animate_nowcasts(nowcasts,linelist_truth, method="bayes.trunc.ddcp",
      control=list(dRange=NULL,anim.dRange=NULL, plot.dRange=NULL,
                   consistent=FALSE, sys.sleep = 1, ylim=NULL,cex.names=0.7,
                   col=c("violetred3","#2171B5","orange","blue","black",
                         "greenyellow")),
                   showLambda=TRUE)
```

Arguments

nowcasts	A list of objects of class stsNC
linelist_truth	True linelist
method	Which method to show (has to be present in the nowcasts)
control	List with control options
showLambda	Boolean indicating whether to show the estimate for the epidemic curve (only applied to bayes.trunc.ddcp)

Value

This function is experimental and not yet fully documented.

Author(s)

M. Höhle

See Also

https://staff.math.su.se/hoehle/blog/2016/07/19/nowCast.html for a worked through example.

stsNewport — *Salmonella Newport cases in Germany 2001-2015*

Description

Reported number of cases of the Salmonella Newport serovar in Germany 2001-2015, by date of disease onset. The slot `control` contains a matrix `reportingTriangle$n` with the reporting triangle as described in Salmon et al. (2015).

Usage

```
data(stsNewport)
```

Format

A `sts` object.

References

Salmon, M., Schumacher, D., Stark, K., Höhle, M. (2015): Bayesian outbreak detection in the presence of reporting delays. Biometrical Journal, 57 (6), 1051-1067.

stsplot — *Plot Methods for Surveillance Time-Series Objects*

Description

This page gives an overview of plot types for 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. The observed term on the left-hand side can also be omitted; it is used by default. 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, 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.
A **ggplot2**-based alternative for this type of plot is provided through an autoplot-method for "sts" objects.

observed ~ time The observations in x are first aggregated over units and the resulting univariate time-series is plotted via the function stsplot_time.

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 (so it actually shows alarm ~ time | unit and this type works as well). 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.

Value

NULL (invisibly). The methods are called for their side-effects.

See Also

the documentation of the individual plot types stsplot_time, stsplot_space, as well as the animate method.

---

stsplot_space *Map of Disease Counts/Incidence accumulated over 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) over 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,
              gpar.missing = list(col = "darkgrey", lty = 2, lwd = 2),
              sp.layout = NULL,
              xlim = bbox(map)[1, ], ylim = bbox(map)[2, ])
```

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 accumulation 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` if `NULL` (default), the map shows the region-specific numbers of cases accumulated over `tps`. For a disease incidence map, `population` can be specified in three ways:

- a numeric vector of population numbers in the `ncol(x)` regions, used to divide the disease counts.
- a matrix of population counts of dimension `dim(x)` (such as `population(x)` in an "sts" object). This will produce the cumulative incidence over `tps` relative to the population at the first time point, i.e., only `population[tps[1],]` is used.
- [if `is(x, "sts")`] a scalar specifying how `population(x)` should be scaled for use as the population matrix, i.e., `population(x)/population` is used. For instance, if `population(x)` contains raw population numbers, `population=1000` would produce the incidence per 1000 inhabitants.

`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 the regions of the `map` are labeled, see `layout.labels`.

`...` further arguments for `spplot`, for example `col = "white"` for white polygon lines.

`at` either a number of levels (default: 10) for the categorization (color-coding) of counts/incidence, or a numeric vector of specific break points, 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 breaks are equally spaced on the square-root scale (equivalent to `sqrt_trans`). Note

that intervals given by `at` are closed on the left and open to the right; if manually specified break points do not cover the data range, further breaks are automatically added at 0 and the maximum (rounded up to 1 significant digit), respectively.

`col.regions` a vector of fill colors, sufficiently long to serve all levels (determined by `at`). "Heat" colors are used by default (`NULL`). A higher-contrast alternative is `hcl.colors(n=10, palette="YlOrRd", rev=TRUE)`.

`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.

`gpar.missing` list of graphical parameters for `sp.polygons` applied to the regions of `map`, which are not part of `x`. Such extra regions won't be plotted if `!is.list(gpar.missing)`.

`sp.layout` optional list of additional layout items, see `spplot`.

`xlim`, `ylim` numeric vectors of length 2 specifying the axis limits.

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)
stsplot_space(measlesWeserEms)  # the same

# cumulative incidence (per 100'000 inhabitants),
# with region labels and white borders
plot(measlesWeserEms, observed ~ unit,
     population = measlesWeserEms@map$POPULATION / 100000,
     labels = list(labels = "GEN", cex = 0.7, font = 3),
     col = "white", lwd = 2,
     sub = "cumulative incidence (per 100'000 inhabitants)")

# incidence in a particular week, manual color breaks, display total
plot(measlesWeserEms, observed ~ unit, tps = 62,
     population = measlesWeserEms@map$POPULATION / 100000,
```

```
     at = c(0, 1, 5),
     total.args = list(x = 0, label = "Overall incidence: "))

# if we had only observed a subset of the regions
plot(measlesWeserEms[,5:11], observed ~ unit,
     gpar.missing = list(col = "gray", lty = 4))
```

---

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=NULL,
             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(),
              dx.upperbound=0L, hookFunc=function(){},
              .hookFuncInheritance=function() {}, ...)

stsplot_alarm(x, lvl=rep(1,ncol(x)),
              xaxis.tickFreq=list("%Q"=atChange),
              xaxis.labelFreq=xaxis.tickFreq, xaxis.labelFormat="%G\n\n%OQ",
              epochsAsDate=x@epochAsDate,
              xlab="time", ylab="", main=NULL,
              outbreak.symbol=list(pch=3, col=3, cex=1, lwd=1),
              alarm.symbol=list(pch=24, col=2, cex=1, lwd=1),
              cex.yaxis=1, ...)
```

Arguments

x	an object of class "sts".
units	optional integer or character vector to select the units (=columns of observed(x)) to plot. The default is to plot all time series. If as.one=FALSE, stsplot_time1 is called for (k in units) with mfrow splitting (see par.list). Note that if

there are too many units, the default mfrow setting might lead to the error "figure margins too large" (meaning that the units do not fit onto a single page).

as.one — logical indicating if all time series should be plotted in a single frame (using matplot).

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 to set graphical parameters before plotting. The mfrow splitting is handled per default. Afterwards, the parameters are reverted to their original values. Use par.list=NULL to disable the internal par call.

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 — arguments for addFormattedXAxis if epochsAsDate=TRUE. Use xaxis.labelFormat=NULL to get a standard x-axis (without date labels).

epochsAsDate — Boolean indicating whether to treat the epochs as Date objects (or to transform them to dates such that the new x-axis formatting is applied). Default: Value of the epochAsDate slot of 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 argument.

alarm.symbol — a list with entries pch, col, cex and lwd specifying the appearance of the alarm 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. Currently ignored by stsplot_alarm.

legend.opts — a list of arguments for the legend. If missing(legend.opts) (i.e., not explicitly specified), the default legend will only be added if the "sts" object contains outbreaks, alarms, or upperbounds. The default legend options are

x "top"

legend c("Infected","Threshold","Outbreak","Alarm")[included]

lty,lwd,pch,col the corresponding graphical settings of the included elements

where individual elements are only `included` in the legend if they are plotted (except for alarms, which are also included if upperbounds exist). To disable the legend, use `legend.opts=NULL`.

`dx.upperbound` horizontal change in the plotting of the upperbound line. Sometimes it can be convenient to offset this line a little for better visibility.

`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.yaxis` The magnification to be used for y-axis annotation.

`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) similar to `plot.survRes`.

Value

`NULL` (invisibly). The functions are called for their side-effects.

Author(s)

Michael Höhle and Sebastian Meyer

See Also

There is an `autoplot`-method, which implements **ggplot2**-based time-series plots of `"sts"` objects.

The `stsplot` help page gives an overview of other types of plots for `"sts"` objects.

Examples

```
data("ha.sts")
print(ha.sts)

plot(ha.sts, type=observed ~ time | unit)  # default multivariate type
plot(ha.sts, units=c("mitt", "pank"))      # selected units
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)

## another multivariate time series example plotted "as.one"
data("measlesDE")
```

```
plot(measlesDE, units=1:2, as.one=TRUE, legend.opts=list(cex=0.8))
## more sophisticated plots are offered by package "xts"
if (requireNamespace("xts", quietly=TRUE))
    plot(xts::as.xts(measlesDE))

## 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")

## Formatting also works for daily data (illustrated by artificial
## outbreak converted to sts object via '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",
     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 (and a replacement version) to extract (or set) the corresponding slot. See the "sts" class documentation.

`stsXtrct` *Subsetting* "sts" *Objects*

Description

The [-method extracts parts of an "sts" object using row (time) and column (unit) indices.

Usage

```
## S4 method for signature 'sts'
x[i, j, ..., drop = FALSE]
```

Arguments

`x`	an object of class `"sts"`.
`i`	optional row index (integer or logical vector).
`j`	optional column index (character, integer, or logical vector).
`drop`	logical: Should subsetting by `j` be applied to the map as well? This requires a *character* index and is disabled by default. It does not affect the other slots: Dimensions are never dropped.
`...`	ignored.

Details

Row indices are used to select a subset of the original time period. The `start` and `epoch` slots of the time series are adjusted accordingly. A warning is issued if an irregular integer sequence is used to extract rows, e.g., `x[c(1,2,4),]`, which could destroy the structure of the time series (`freq`).

Column indices work as usual when indexing matrices, so may select units by name, position or a vector of booleans. When subsetting columns, population fractions are recomputed if and only if `x` is no `multinomialTS` and already contains population fractions.

`NA` indices are not supported, negative indices are.

Note that a `[<-` method (i.e., subassignment) is not implemented.

Value

an object of class `"sts"`.

Examples

```
data("ha.sts")

# Show a (subset of a) single time series
plot(ha.sts[,7])
plot(ha.sts[year(ha.sts)==2006, 7])

# Map a single time point
plot(ha.sts[5*52+26,], type=observed~unit)
plot(ha.sts,           type=observed~unit, tps=5*52+26) # same -> ?stsplot_space

# Restrict the data (and the map) to a subset of the districts
plot(ha.sts[,c("pank","lich")],           type=observed~unit, labels=TRUE)
plot(ha.sts[,c("pank","lich"),drop=TRUE], type=observed~unit, labels=TRUE)
```

`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:

stsTickFactors: A named vector containing tick sizes for the `"sts"` x-axis relative to `par("tcl")`. Each entry contains the size at `strptime` formatting strings. See the help on `stsplot_time1` for 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 queried before running cumbersome computations in help file examples. For `interactive()` sessions, this option defaults to `TRUE`. Otherwise, long examples will only be run if the environment variable `_R_SURVEILLANCE_ALL_EXAMPLES_` is set (to any value different from `""`) when loading the **surveillance** package. This is to avoid long computations during (daily) CRAN checks.

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.

Examples

```
surveillance.options()
```

`tidy.sts` *Convert an "sts" Object to a Data Frame*

Description

The `as.data.frame` method for "`sts`" objects `cbind`s the following slots into a *wide* data frame (one row per observation time): `observed`, `epoch`, `state`, `alarm`, `upperbound`, `populationFrac` (as `population`), `freq`, and `epochInPeriod`.

The `tidy.sts` function additionally `reshape`s multivariate time series into the *long* format (one row per time $\times$ unit), adding a `unit` column as well as the useful `epoch` transformations `year`, `epochInYear`, and `date`. The `tidy` format is particularly useful for regression models or plotting functions.

Usage

```
tidy.sts(x, ...)

## S3 method for class 'sts'
as.data.frame(x, ..., tidy = FALSE, as.Date = x@epochAsDate)
```

Arguments

`x`	an object of class "`sts`".
`...`	unused (argument of the generic).
`tidy`	logical indicating if `tidy.sts` should be called.
`as.Date`	logical indicating if `epoch` should be of class "`Date`". Ignored in the `tidy` case, where `epoch` remains numeric and an extra column `date` is added.

Value

a data frame combining slots of the "`sts`" input.

For `as.data.frame`, the number of rows equals the number of time points, `nrow(x)`, and there are two special columns:

`freq`	for weekly or daily data and if `x@epochAsDate` is `TRUE`, the number of samples in the current year, otherwise `x@freq`.

epochInPeriod — the epoch fraction within the current year, epochInYear(x)/freq.

For tidy.sts, the resulting data frame has one row for each combination of epoch and unit, the latter a factor with levels the colnames(x). It contains the above special columns plus the following extra time variables:

year — numeric, from year(x).

epochInYear — numeric, the day/week/month number (or, more generally, "cycle" within x@freq), from epochInYear(x).

date — "Date", from epoch(x, as.Date=TRUE) if possible, otherwise NA (for non-standard frequency(x) if x@epochAsDate is false).

Author(s)

Sebastian Meyer

Examples

```
data("momo")
momo

momolong <- tidy.sts(momo)
head(momolong, 3)
str(momolong)

momowide <- as.data.frame(momo)
str(momowide, vec.len = 2)  # unit-specific columns, like "sts" matrix slots
stopifnot(nrow(momowide) == nrow(momo))

## tidy.sts() is the same as as.data.frame(<sts>, tidy = TRUE)
stopifnot(identical(as.data.frame(momo, tidy = TRUE), momolong))
```

---

toLatex.sts — toLatex-*Method for* "sts" *Objects*

Description

Convert "sts" objects to a character vector with LaTeX markup using **xtable**.

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)


toLatex(salm, sanitize.text.function=identity, comment=FALSE)
```

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. The implementation is illustrated in Meyer et al. (2017, Section 4), see vignette("twinSIR").

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 the model specification are given below.
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 the selected subset of the data, which is always restricted to atRiskY == 1 rows. 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 – only a 1st order differences penalty matrix as in Fahrmeir and Lang (2001) is 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")` in package **survival**.

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 convergence:

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 multiplication by the parameter vector β). With large covariate values, 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"`, which 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.

Meyer, S., Held, L. and Höhle, M. (2017): Spatio-temporal analysis of epidemic phenomena using the R package **surveillance**. *Journal of Statistical Software*, **77** (11), 1-55. doi:10.18637/jss.v077.i11

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("hagelloch")
summary(hagelloch)

# simple model with an overall constant baseline hazard rate
fit1 <- twinSIR(~ household + cox(AGE), data = hagelloch)
fit1
summary(fit1)   # see also help("summary.twinSIR")
plot(fit1)      # see also help("plot.twinSIR")
checkResidualProcess(fit1)   # could be better

# fit a piecewise constant baseline hazard rate with 3 intervals using
# _un_penalized ML and estimated coefs from fit1 as starting values,
# without tracing
fit2 <- twinSIR(~ household, data = hagelloch, nIntervals = 3,
                optim.args = list(par = coef(fit1)[c(1,2,2,2)],
                                  control = list(trace = 0)))
summary(fit2)

# fit a piecewise constant baseline hazard rate with 7 intervals
# using _penalized_ ML
fit3 <- twinSIR(~ household, data = hagelloch, nIntervals = 7,
                lambda.smooth = 0.1, penalty = 1)
summary(fit3)
checkResidualProcess(fit3)

# plot the estimated log-baseline levels
plot(x=fit2$intervals, y=coef(fit2)[c(2,2:4)], type="S", ylim=c(-6, -1))
lines(x=fit3$intervals, y=coef(fit3)[c(2,2:8)], type="S", col=2)
legend("right", legend=c("unpenalized 3", "penalized 7"), lty=1, col=1:2, bty="n")


## special use case: fit the model to a subset of the events only,
## while preserving epidemic contributions from the remainder
## (maybe some buffer area nodes)
```

```
fit_subset <- twinSIR(~ household, data = hagelloch, subset = CL=="preschool")
summary(fit_subset)
```

---

twinSIR_cox *Identify Endemic Components in an Intensity Model*

Description

cox() terms are marked "special" in formulae of the functions twinSIR and simEpidata to be interpreted as endemic components, i.e., variables acting multiplicatively on the baseline infection intensity.

Usage

```
cox(x)
```

Arguments

x	a variable from the modelled epidata.

Value

the trivial identity(x): it simply returns its argument.

---

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. The implementation is illustrated in Meyer et al. (2017, Section 4), see vignette("twinSIR").

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 method, 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

Meyer, S., Held, L. and Höhle, M. (2017): Spatio-temporal analysis of epidemic phenomena using the R package **surveillance**. *Journal of Statistical Software*, **77** (11), 1-55. doi:10.18637/jss.v077.i11

See Also

twinSIR for a description of the intensity model, and simulate.twinSIR for the simulation of epidemic data according to a twinSIR specification.

Examples

```
data("hagelloch")
plot(hagelloch)

# a simplistic twinSIR model
fit <- twinSIR(~ household, data = hagelloch)

# overall total intensity
plot(fit, which = "total")

# overall epidemic proportion
epi <- plot(fit, which = "epidemic", ylim = c(0, 1))
head(epi)
# add overall endemic proportion = 1 - epidemic proportion
ende <- plot(fit, which = "endemic", add = TRUE, col = 2)
legend("topleft", legend = "endemic proportion", lty = 1, col = 2, bty = "n")

# individual intensities
tmp <- plot(fit, 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 = 1:2, lty = 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 determined 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("hagelloch")

# a simplistic twinSIR model
fit <- twinSIR(~ household + cox(AGE), data = hagelloch)

coef(fit)
vcov(fit)
logLik(fit)

summary(fit, correlation = TRUE, symbolic.cor = TRUE)

# AIC or OSAIC
AIC(fit)
AIC(fit, one.sided = FALSE)
extractAIC(fit)
extractAIC(fit, one.sided = FALSE)

# comparing models via AIC
fit2 <- update(fit, nIntervals = 2)
AIC(fit, fit2)   # the 2nd column should 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! There is a simple `plot`-method for the result.

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

a list with profile log-likelihood evaluations on the grid and highest likelihood and Wald confidence intervals. The argument `profile` is also returned. The result has class "profile.twinSIR", for which a simple (undocumented) `plot`-method is available.

Author(s)

Michael Höhle and Sebastian Meyer

Examples

```
data("hagelloch")
fit <- twinSIR(~ household, data = hagelloch)
gridsize <- if (interactive()) 35 else 5  # for fast tests
prof <- profile(fit, list(c(1, NA, NA, gridsize)))
prof$ci.hl
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 split 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 Euclidean 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. Note that `end` should be set to a finite value in these cases.

`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 `cat`ed. 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=c(rep(1,3),rep(0,n-3)), # 3 initially infectious
                  start=rep(0,n), stop=rep(Inf,n))

## Simulate an SIR epidemic in this population
set.seed(123)
infPeriods <- setNames(c(1:3/10, rexp(n-3, rate=1)), 1:n)
epi <- simEpidata(
    cbind(start,stop) ~ cox(gender), data = pop,
    id.col = "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) infPeriods[ids],
    ##remPeriod = function(ids) rexp(length(ids), rate=0.1), end = 30   # -> SIRS
)

## extract event times by id
head(summary(epi)$byID)

## Plot the numbers of susceptible, infectious and removed individuals
plot(epi)


## load the 1861 Hagelloch measles epidemic
data("hagelloch")
summary(hagelloch)
plot(hagelloch)

## fit a simplistic twinSIR model
fit <- twinSIR(~ household, data = hagelloch)

## simulate a new epidemic from the above model
## with simulation period = observation period, re-using observed infPeriods
sim1 <- simulate(fit, data = hagelloch)
plot(sim1)

## check if we find similar parameters in the simulated epidemic
fitsim1 <- update(fit, data = sim1)
cbind(base = coef(fit), new = coef(fitsim1))


if (surveillance.options("allExamples")) {

## simulate only 10 days, using random infPeriods ~ Exp(0.1)
sim2 <- simulate(fit, data = hagelloch, seed = 2, end = 10,
    infPeriod = function(ids) rexp(length(ids), rate = 0.1))
plot(sim2)
```

```
## simulate from a different model with manually specified parameters
set.seed(321)
simepi <- simEpidata(~ cox(AGE), data = hagelloch,
    beta = c(0.1), h0 = -4, alpha = c(household = 0.05),
    f = list(household = function(u) u == 0),
    infPeriod = function(ids) rexp(length(ids), rate=1/8))
plot(simepi)
intensityplot(simepi)

## see if we correctly estimate the parameters
fitsimepi <- twinSIR(~ cox(AGE) + household, data = simepi)
cbind(true = c(0.05, -4, 0.1), est = coef(fitsimepi), confint(fitsimepi))

}
```

---

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. The implementation is illustrated in Meyer et al. (2017, Section 3), see vignette("twinstim").

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,
         epilink = "log", 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, more commonly, generated by a predefined interaction function such as `siaf.gaussian` as in Meyer et al. (2012) or `siaf.powerlaw` as in Meyer and Held (2014). The latter requires unique event locations, possibly after random tie-breaking (`untie`) or imputation of interval-censored locations. `siaf.exponential` is a simpler alternative.
- 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 occurred during (-Inf;t0] are regarded as part of the prehistory H_0 of the process. Only events that occurred 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.

`epilink` a character string determining the link function to be used for the `epidemic` linear predictor of event marks. By default, the log-link is used. The experimental alternative `epilink = "identity"` (for use by `epitest`) does not guarantee the force of infection to be positive. If this leads to a negative total intensity (endemic + epidemic), the point process is not well defined (the log-likelihood will be `NaN`).

`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 control the accuracy of the cubature routines 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(F.adaptive = TRUE)` 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`. However, the default version `siaf.gaussian()` as well as `siaf.powerlaw()` and friends use `polyCub.iso` and thus accept control arguments for the standard `integrate` routine (such as `rel.tol`) via `control.siaf$F` and `control.siaf$Deriv`.
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, crude estimates will be used for the endemic intercept and the Gaussian kernel, -9 for the epidemic intercept, and zeroes for the remaining parameters. 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, `traceing` 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 likelihood 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 appropriately: 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).

Similarly, 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 indicating if the model environment should be kept with the result, which is required for intensityplots and R0(..., trimmed = FALSE). Specifically, if model=TRUE, the return value will have the evaluation environment set as its environment, and the returned functions element will contain 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). Note that fitted objects with a model environment might consume quite a lot of memory since they contain the data.

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 summary output can be converted via corresponding xtable or toLatex methods. Furthermore, the usual accessor methods are implemented, including coef, vcov, logLik, residuals, and update. Additional functionality is provided by the R0 and simulate methods.

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.
- `xlevels` a record of the levels of the factors used in fitting.
- `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 https://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:10.1111/j.15410420.2011.01684.x

Meyer, S. and Held, L. (2014): Power-law models for infectious disease spread. *The Annals of Applied Statistics*, **8** (3), 1612-1639. doi:10.1214/14AOAS743

Meyer, S., Held, L. and Höhle, M. (2017): Spatio-temporal analysis of epidemic phenomena using the R package **surveillance**. *Journal of Statistical Software*, **77** (11), 1-55. doi:10.18637/jss.v077.i11

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

`vignette("twinstim")`

There is a `simulate.twinstim` method, which simulates the point process based on the fitted `twinstim`.

A discrete-space alternative is offered by the `twinSIR` modelling framework.

Examples

```
# Load invasive meningococcal disease data
data("imdepi")


### first, fit a simple 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)
)

## look at the model summary
summary(m_noepi)

## there is no evidence for a type-dependent endemic intercept (LR test)
m_noepi_type <- update(m_noepi, endemic = ~(1|type) + .)
pchisq(2*c(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, model=TRUE)

## summarize the model fit
summary(m0, correlation = TRUE, symbolic.cor = TRUE)

## the default confint-method can be used for Wald-CI's
confint(m0, level=0.95)

## same "untrimmed" R0 for every event (simple epidemic intercept model)
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)
# easier, with CI and serial correlation:
checkResidualProcess(m0)
}

## Not run:
  ## NB: in contrast to nlminb(), optim's BFGS would miss the
```

```
  ##     likelihood maximum wrt the epidemic intercept
  m0_BFGS <- update(m_noepi, epidemic=~1, optim.args = list(method="BFGS"))
  format(cbind(nlminb=coef(m0), BFGS=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)


### an epidemic-only model?
## for a purely epidemic model, all events must have potential source events
## (otherwise the intensity at the observed event would be 0)

## let's focus on the C-type for this example
imdepiC <- subset(imdepi, type == "C")
table(summary(imdepiC)$nSources)
## 106 events have no prior, close events (in terms of eps.s and eps.t)
try(twinstim(epidemic = ~1, data = imdepiC))  # detects this problem
## let's assume spatially unbounded interaction
imdepiC_infeps <- update(imdepiC, eps.s = Inf)
s <- summary(imdepiC_infeps)
table(s$nSources)
## for 11 events, there is no prior event within eps.t = 30 days
## (which is certainly true for the first event)
plot(s$counter, main = "Number of infectious individuals over time (eps.t = 30)")
rug(imdepiC_infeps$events$time)
rug(imdepiC_infeps$events$time[s$nSources == 0], col = 2, lwd = 3)
## An endemic component would catch such events (from unobserved sources),
## otherwise a longer infectious period would need to be assumed and
## for the first event to happen, a prehistory is required (e.g., t0 = 31).
## As an example, we fit the data only until T = 638 (all events have ancestors)
m_epi <- twinstim(epidemic = ~1, data = imdepiC_infeps, t0 = 31, T = 638)
summary(m_epi)


if (surveillance.options("allExamples")) withAutoprint({

### full model with interaction functions (time-consuming)
## estimate an exponential temporal decay of infectivity
m1_tiaf <- update(m0, tiaf=tiaf.exponential())
plot(m1_tiaf, "tiaf", scaled=FALSE)

## estimate a step function for spatial interaction
summary(sourceDists <- getSourceDists(imdepi, "space"))
(knots <- quantile(sourceDists, c(5,10,20,40)/100))
m1_fstep <- update(m0, siaf=knots)
plot(m1_fstep, "siaf", scaled=FALSE)
rug(sourceDists, ticksize=0.02)
```

```
## estimate a continuously decreasing spatial interaction function,
## here we use the kernel of an isotropic bivariate Gaussian
m1 <- update(m0, siaf = siaf.gaussian())
AIC(m_noepi, m0, m1_fstep, m1)
summary(m1)  # e.siaf.1 is log(sigma), no test for H0: log(sigma) = 0
exp(confint(m1, "e.siaf.1"))  # a confidence interval for sigma
plot(m1, "siaf", scaled=FALSE)
## alternative: siaf.powerlaw() with eps.s=Inf and untie()d data,
##              see vignette("twinstim")

## 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_epitest` *Permutation Test for Space-Time Interaction in* "twinstim"

Description

The function `epitest` takes a "twinstim" model and tests if the spatio-temporal interaction invoked by the epidemic model component is statistically significant. The test only works for simple epidemic models, where `epidemic = ~1` (no additional parameters for event-specific infectivity), and requires the non-canonical `epilink="identity"` (see `twinstim`). A permutation test is performed by default, which is only valid if the endemic intensity is space-time separable. The approach is described in detail in Meyer et al. (2016), where it is also compared to alternative global tests for clustering such as the `knox` test.

Usage

```
epitest(model, data, tiles, method = "time", B = 199,
        eps.s = NULL, eps.t = NULL, fixed = NULL,
        verbose = TRUE, compress = FALSE, ...)

## S3 method for class 'epitest'
coef(object, which = c("m1", "m0"), ...)
## S3 method for class 'epitest'
plot(x, teststat = c("simpleR0", "D"), ...)
```

Arguments

`model` a simple epidemic "`twinstim`" with `epidemic = ~1`, fitted using the non-canonical `epilink="identity"`. Note that the permutation test is only valid for models with a space-time separable endemic intensity, where covariates vary either in space or time but not both.

data	an object of class "epidataCS", the data to which the model was fitted.
tiles	(only used by method = "simulate") a "SpatialPolygons" representation of the tiles in data$stgrid.
method	one of the following character strings specifying the test method: "LRT": a simple likelihood ratio test of the epidemic model against the corresponding endemic-only model, "time"/"space": a Monte Carlo permutation test where the null distribution is obtained by relabeling time points or locations, respectively (using permute.epidataCS). "simulate": obtain the null distribution of the test statistic by simulations from the endemic-only model (using simEndemicEvents).
B	the number of permutations for the Monte Carlo approach. The default number is rather low; if computationally feasible, B = 999 is more appropriate. Note that this determines the "resolution" of the p-value: the smallest attainable p-value is 1/(B+1).
eps.s, eps.t	arguments for simpleR0.
fixed	optional character vector naming parameters to fix at their original value when re-fitting the model on permuted data. The special value fixed = TRUE means to fix all epidemic parameters but the intercept.
verbose	the amount of tracing in the range 0:3. Set to 0 (or FALSE) for no output, 1 (or TRUE, the default) for a progress bar, 2 for the test statistics resulting from each permutation, and to 3 for additional tracing of the log-likelihood maximization in each permutation (not useful if parallelized). Tracing does not work if permutations are parallelized using clusters. See plapply for other choices.
compress	logical indicating if the nobs-dependent elements "fitted", "fittedComponents", and "R0" should be dropped from the permutation-based model fits. Not keeping these elements saves a lot of memory especially with a large number of events. Note, however, that the returned permfits then no longer are fully valid "twinstim" objects (but most methods will still work).
...	further arguments for plapply to configure parallel operation, i.e., .parallel as well as .seed to make the results reproducible. For the plot-method, further arguments passed to truehist. Ignored by the coef-method.
object, x	an object of class "epitest" as returned by epitest.
which	a character string indicating either the full ("m1", default) or the endemic-only ("m0") model.
teststat	a character string determining the test statistic to plot, either "simpleR0" or "D" (twice the log-likelihood difference of the models).

Details

This space-time interaction test is limited to models with epidemic = ~1, since covariate effects are not identifiable under the null hypothesis of no space-time interaction. Estimating a rich epidemic model based on permuted data will most likely result in singular convergence. A similar issue might arise when the model employs parametric interaction functions, in which case fixed=TRUE can be used. For further details see Meyer et al. (2016).

The test statistic is the reproduction number `simpleR0`. A likelihood ratio test of the supplied epidemic model against the corresponding endemic-only model is also available. By default, the null distribution of the test statistic under no space-time interaction is obtained by a Monte Carlo permutation approach (via `permute.epidataCS`) and therefore relies on a space-time separable endemic model component.

The `plot`-method shows a `truehist` of the simulated null distribution together with the observed value. The `coef`-method extracts the parameter estimates from the `B` `permfits` (by default for the full model `which = "m1"`).

Value

a list (inheriting from `"htest"`) with the following components:

`method`	a character string indicating the type of test performed.
`data.name`	a character string giving the supplied `data` and `model` arguments.
`statistic`	the observed test statistic.
`parameter`	the (effective) number of permutations used to calculate the p-value (only those with convergent fits are used).
`p.value`	the p-value for the test. For the `methods` involving resampling under the null (`method != "LRT"`), it is based on the subset of convergent fits only and the p-value from the simple LRT is attached as an attribute `"LRT"`.

In addition, if `method != "LRT"`, the result will have the following elements:

`permfits`	the list of model fits (endemic-only and epidemic) from the `B` permutations.
`permstats`	a data frame with `B` rows and the columns `"l0"` (log-likelihood of the endemic-only model `m0`), `"l1"` (log-likelihood of the epidemic model `m1`), `"D"` (twice their difference), `"simpleR0"` (the results of `simpleR0(m1, eps.s, eps.t)`), and `"converged"` (a boolean indicator if both models converged).

The `plot`-method invisibly returns `NULL`. The `coef`-method returns the `B` x `length(coef(model))` matrix of parameter estimates.

Author(s)

Sebastian Meyer

References

Meyer, S., Warnke, I., Rössler, W. and Held, L. (2016): Model-based testing for space-time interaction using point processes: An application to psychiatric hospital admissions in an urban area. *Spatial and Spatio-temporal Epidemiology*, **17**, 15-25. doi:10.1016/j.sste.2016.03.002. Eprint: https://arxiv.org/abs/1512.09052.

See Also

`permute.epidataCS`, `knox`

Examples

```
data("imdepi", "imdepifit")

## test for space-time interaction of the B-cases
## assuming spatial interaction to be constant within 50 km
imdepiB50 <- update(subset(imdepi, type == "B"), eps.s = 50)
imdfitB50 <- update(imdepifit, data = imdepiB50, subset = NULL,
                    epidemic = ~1, epilink = "identity", siaf = NULL,
                    start = c("e.(Intercept)" = 0))

## simple likelihood ratio test
epitest(imdfitB50, imdepiB50, method = "LRT")

## permutation test
et <- epitest(imdfitB50, imdepiB50,
              B = 5,        # CAVE: limited here for speed
              verbose = 2,  # (tracing does not work on Windows
              .seed = 1, .parallel = 1)       # if parallelized)
et
plot(et)

## summary of parameter estimates under permutation
summary(coef(et, which = "m1"))
```

---

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. Own such functions can be specified (see siaf and tiaf, respectively), but the package already predefines some common dispersal kernels returned by the constructor functions documented here. See Meyer and Held (2014) for various spatial interaction functions, and Meyer et al. (2017, Section 3, available as vignette("twinstim")) for an illustration of the implementation.

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 = FALSE, F.method = "iso",
              effRangeMult = 6, validpars = NULL)
siaf.exponential(nTypes = 1, validpars = NULL, engine = "C")
siaf.powerlaw(nTypes = 1, validpars = NULL, engine = "C")
siaf.powerlaw1(nTypes = 1, validpars = NULL, sigma = 1)
```

```
siaf.powerlawL(nTypes = 1, validpars = NULL, engine = "C")
siaf.student(nTypes = 1, validpars = NULL, engine = "C")

# 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 $[a, b)$.
An initial choice of knots could be based on quantiles of the observed distances between events and their potential source events. For instance, an identifiable spatial step function could be `siaf.step(quantile(getSourceDists(myepi, "space"), c(1,2,4)/10))`, where `myepi` is the `"epidataCS"` data to be modelled.

`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(F.adaptive = TRUE)`, `tiaf.step`, and `tiaf.exponential`.

`logsd, density` logicals affecting the parametrization of the Gaussian kernel. Settings different from the defaults are deprecated. The default is to use only the kernel of the bivariate, isotropic normal distribution (`density=FALSE`, see Details below), parametrized with the log-standard deviation (`logsd=TRUE`) to avoid constrained optimisation (L-BFGS-B) or `validpars`.
The power-law kernels always employ the log-scale for their scale and shape parameters.

`F.adaptive, F.method`
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`, the `F.method` argument determines which `polyCub` method to use in `siaf$F`. The accuracy (controlled via, e.g., `nGQ`,

rel.tol, or eps, depending on the cubature method) can then be adjusted 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 the 6σ region around the event is considered as the relevant integration domain 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.

engine — character string specifying the implementation to use. Prior to **surveillance** 0.14.0, the intrfr functions for polyCub.iso were evaluated in R (and this implementation is available via engine = "R"). The new C-implementation, 'LinkingTo' the newly exported polyCub_iso C-implementation in **polyCub** 0.6.0, is considerably faster.

sigma — Fixed value of σ for the one-parameter power-law kernel.

Details

Evaluation of twinstim's likelihood involves cubature of the spatial interaction function over polygonal domains. Various approaches have been compared by Meyer (2010, Section 3.2) and a new efficient method, which takes advantage of the assumed isotropy, has been proposed by Meyer and Held (2014, Supplement B, Section 2) for evaluation of the power-law kernels. These cubature methods are available in the dedicated R package **polyCub** and used by the kernels implemented in **surveillance**.

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 (deprecated), 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 $\sigma_\kappa^2 I_2$. The standard deviation is optimized on the log-scale (logsd = TRUE, not doing so is deprecated).

siaf.exponential: $f(s) = exp(-||s||/sigma)$
The scale parameter $sigma$ is estimated on the log-scale, i.e., $\sigma = \exp(\tilde{\sigma})$, and $\tilde{\sigma}$ is the actual model parameter.

`siaf.powerlaw`: $f(s) = (||s|| + \sigma)^{-d}$
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. If a power-law kernel is not identifiable for the dataset at hand, the exponential kernel or a lagged power law are useful alternatives.

`siaf.powerlaw1`: $f(s) = (||s|| + 1)^{-d}$,
i.e., `siaf.powerlaw` with fixed $\sigma = 1$. A different fixed value for $sigma$ can be specified via the `sigma` argument of `siaf.powerlaw1`. The decay parameter d is estimated on the log-scale.

`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 workarounds 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. (2010): Spatio-Temporal Infectious Disease Epidemiology based on Point Processes. Master's Thesis, Ludwig-Maximilians-Universität München.
Available as https://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:10.1111/j.15410420.2011.01684.x

Meyer, S. and Held, L. (2014): Power-law models for infectious disease spread. *The Annals of Applied Statistics*, **8** (3), 1612-1639. doi:10.1214/14AOAS743

Meyer, S., Held, L. and Höhle, M. (2017): Spatio-temporal analysis of epidemic phenomena using the R package **surveillance**. *Journal of Statistical Software*, **77** (11), 1-55. doi:10.18637/jss.v077.i11

See Also

twinstim, siaf, tiaf, and package **polyCub** for the involved cubature methods.

Examples

```
# constant temporal dispersal
tiaf.constant()
# type-specific step function kernel
tiaf.step(c(3,7), maxRange=14, nTypes=2)$npars  # 4 parameters (2 per type)
# exponential temporal decay function
tiaf.exponential()$g

# Type-dependent Gaussian spatial interaction function using an adaptive
# two-dimensional midpoint-rule to integrate it over polygonal domains
str(siaf.gaussian(2, F.adaptive=TRUE))

# Single-type Gaussian spatial interaction function (using polyCub.iso)
siaf.gaussian()$f

# Exponential kernel
siaf.exponential()$f

# Power-law kernel
siaf.powerlaw()$f

# Power-law kernel with fixed sigma = 1
siaf.powerlaw1()$f

# "lagged" power-law
siaf.powerlawL()$f

# (reparametrized) t-kernel
siaf.student()$f

# step function kernel
siaf.step(c(10,20,50), maxRange=100)$f
```

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. The implementation is illustrated in Meyer et al. (2017, Section 3), see vignette("twinstim").

Usage

```
iafplot(object, which = c("siaf", "tiaf"), types = NULL,
        scaled = c("intercept", "standardized", "no"), 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 $f(x)$ as a function of the distance x, and "tiaf" for the temporal interaction function $g(t)$.

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 — character string determining if/how the the interaction function should be scaled. Possible choices are:

"intercept": multiplication by the epidemic intercept.

"standardized": division by the value at 0 distance such that the function starts at 1.

"no": no scaling.

The first one is the default and required for the comparison of estimated interaction functions from different models. For backward compatibility, scaled can also be a boolean, where TRUE refers to "intercept" scaling and FALSE to "no" scaling.

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 the value of the (scaled) interaction function at $x = 0$. 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 diagonal 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:10.1111/j.15410420.2011.01684.x

Meyer, S., Held, L. and Höhle, M. (2017): Spatio-temporal analysis of epidemic phenomena using the R package **surveillance**. *Journal of Statistical Software*, **77** (11), 1-55. doi:10.18637/jss.v077.i11

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, or its epidemic or endemic components/proportions 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 = "epidemic proportion",
    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" (default), "endemic proportion", or "total intensity" (partially matched), or "epidemic intensity" or "endemic intensity". Determines whether to plot the total or component-specific intensities or their proportions as a function of aggregate.
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 approximate number of points of a grid constructed on the bounding box of tiles. 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). The default border colour of the polygons is "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 is 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. `intensity.twinstim` returns a list containing such functions for the endemic and epidemic intensity (but these are not vectorized).

If `plot = TRUE` and `aggregate = "space"`, the `trellis.object` of the spatial plot is returned.

Author(s)

Sebastian Meyer

See Also

`plot.twinstim`, which calls `intensityplot.twinstim`.

Examples

```
data("imdepi", "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 (default) by event
intensityplot(imdepifit, tgrid=500, types=1)
intensityplot(imdepifit, tgrid=500, types=2,
              add=TRUE, col=2, rug.opts=list(col=2))
legend("topright", legend=levels(imdepi$events$type), lty=1, col=1:2,
       title = "event type")

## endemic and total intensity in one plot
intensityplot(imdepifit, which="total intensity", tgrid=501, lwd=2,
              ylab="intensity")
intensityplot(imdepifit, which="endemic intensity", tgrid=501, lwd=2,
              add=TRUE, col=2, rug.opts=NULL)
text(2500, 0.36, labels="total", col=1, pos=2, font=2)
text(2500, 0.08, labels="endemic", col=2, pos=2, font=2)


## spatial shape of the intensity (aggregated over time)

# need a map of the 'stgrid' tiles, here 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,
              col.regions=rev(heat.colors(100)))

if (surveillance.options("allExamples")) {
  # epidemic proportion by type
  maps_epiprop <- lapply(1:2, function (type) {
      intensityplot(imdepifit, which="epidemic", aggregate="space",
                    types=type, tiles=districtsD, sgrid=1000,
                    main=rownames(imdepifit$qmatrix)[type],
                    scales=list(draw=FALSE), at=seq(0,1,by=0.1),
                    col.regions=rev(hcl.colors(10,"YlOrRd")),
                    par.settings=list(par.title.text=list(cex=1)))
  })
  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, runtime = FALSE, ...)

## S3 method for class 'twinstim'
coeflist(x, ...)
## 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.

`runtime` — logical. If `TRUE`, the summary additionally includes the time elapsed and the number of log-likelihood and score function evaluations during model fitting.

`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 separator `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**. Note, however, that there is the useful `coeflist` method to list the coefficients by model component.

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 prehistory), 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 (in a single vector or listed by model component)
coef(imdepifit)
coeflist(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)

# produce a summary with parameter correlations and runtime information
(s <- summary(imdepifit, correlation=TRUE, symbolic.cor=TRUE, runtime=TRUE))

# create LaTeX code of coefficient table
if (requireNamespace("xtable"))
  toLatex(s, digits=2)

# or using the xtable-method (which produces rate ratios)
if (requireNamespace("xtable"))
  xtable::xtable(s)
```

---

twinstim_plot *Plots for a Fitted* twinstim

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 (e.g., "siaf") and intensityplot.twinstim (e.g., "epidemic proportion", partially matched).
...	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

`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, maxit = 100, trace = 1),
        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 – not implemented yet – highest likelihood and Wald confidence intervals. The argument `profile` is also returned.

Author(s)

Michael Höhle

Examples

```
# profiling takes a while
## Not run:
#Load the twinstim model fitted to the IMD data
data("imdepi", "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(reltol=0.1, REPORT=1), 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 upper bound 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_simEndemicEvents`

Quick Simulation from an Endemic-Only `twinstim`

Description

In *endemic-only* `twinstim` models, the conditional intensity is a piecewise constant function independent from the history of the process. This allows for a much more efficient simulation algorithm than via Ogata's modified thinning as in the general `simulate.twinstim` method.

Usage

```
simEndemicEvents(object, tiles)
```

Arguments

`object` an object of class "`twinstim`" (with the `model` component retained; otherwise try `object <- update(object, model = TRUE)`).

`tiles` an object inheriting from "`SpatialPolygons`", which represents the tiles of the original data's `stgrid` (see, e.g., `levels(environment(object)$gridTiles)`).

Value

a "SpatialPointsDataFrame"

Author(s)

Sebastian Meyer

See Also

the general simulation method simulate.twinstim

Examples

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

## Fit an endemic-only twinstim()
m_noepi <- update(imdepifit, epidemic = ~0, siaf = NULL, model = TRUE,
                  T = 120)  # using a restricted time range, for speed

## Simulate events from the above endemic model
set.seed(1)
s1 <- simEndemicEvents(m_noepi, tiles = districtsD)
class(s1)  # just a "SpatialPointsDataFrame"
summary(s1@data)
plot(imdepi$W, lwd = 2, asp = 1)
plot(s1, col = s1$type, cex = 0.5, add = TRUE)

## Compare with the generic simulation method (slower)
s0 <- simulate(m_noepi, seed = 1, data = imdepi, tiles = districtsD)
class(s0)  # gives a full "simEpidataCS" with several methods applicable
methods(class = "epidataCS")
plot(s0, "time")
plot(s0, "space", points.args = list(pch = 3), lwd = 2)
```

---

twinstim_simulation *Simulation of a Self-Exciting Spatio-Temporal Point Process*

Description

The function simEpidataCS simulates events of a self-exciting 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,
    epilink = "log", 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,
    newcoef = NULL, 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 argument) and location (2nd argument) 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.

epilink
: a character string determining the link function to be used for the epidemic linear predictor of event marks. By default, the log-link is used. The experimental alternative is epilink = "identity". Note that the identity link does not guarantee the force of infection to be positive. If this leads to a negative total intensity (endemic + epidemic), the point process is not well defined and simulation cannot proceed.

t0
: events having occurred during (-Inf;t0] are regarded as part of the prehistory H_0 of the process. For simEpidataCS, by default and also if t0=NULL, the beginning of stgrid is used as t0. For the simulate.twinstim method, NULL means to use the fitted time range 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`.

`newcoef` an optional named numeric vector of (a subset of) parameters to replace the original point estimates in `coef(object)`. Elements which do not match any model parameter by name are silently ignored. The `newcoefs` may also be supplied in a list following the same conventions as for the `start` argument in `twinstim`.

`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"`: `bbox`, `timeRange`, `formula`, `coefficients`, `npars`, `control.siaf`, `call`, `runtime`. It has corresponding `coeflist`, `residuals`, `R0`, and `intensityplot` methods.

The `simulate.twinstim` method has some additional *attributes* set on its result: `call`, `seed`, and `runtime`. If `nsim > 1`, it returns an object of class `"simEpidataCSlist"`, the form of which depends

on the value of `simplify` (which is stored as an attribute `simplified`): 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. You are advised to sacrifice some shape details for speed by reducing the polygon complexity, for example via the `mapshaper` JavaScript library wrapped by the R package **rmapshaper**, or via `simplify.owin`.

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

Meyer, S., Elias, J. and Höhle, M. (2012): A space-time conditional intensity model for invasive meningococcal disease occurrence. *Biometrics*, **68**, 607-616. doi:10.1111/j.15410420.2011.01684.x

See Also

The function `simEndemicEvents` is a faster alternative for endemic-only models, only returning a `"SpatialPointsDataFrame"` of simulated events.

The `plot.epidataCS` and `animate.epidataCS` methods for plotting and animating continuous-space epidemic data, respectively, also work for simulated epidemics (by inheritance), and `twinstim` can be used to fit spatio-temporal conditional intensity models also to simulated data.

Examples

```
data("imdepi", "imdepifit")

## simulation needs the district polygons ('tiles')
load(system.file("shapes", "districtsD.RData", package="surveillance"))
if (surveillance.options("allExamples")) {
  plot(districtsD)
  plot(stateD, add=TRUE, border=2, lwd=2)
}

### simulate from the fitted model, only over the first 90 days (for speed)
```

```
nsim <- 10

sims <- simulate(imdepifit, nsim=nsim, seed=1, data=imdepi, T=90,
                 tiles=districtsD, simplify=TRUE)

sims

## plot event times for a random selection of 4 simulations
plot(sims, aggregate="time")

## extract the second realization -> object of class "simEpidataCS"
sim2 <- sims[[2]]
summary(sim2)
plot(sim2, aggregate="space")

## extract _cumulative_ number of events (simulated vs. observed)
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=adjustcolor(c(2,4), alpha.f=0.5)))


## Not run:
### Experimental code to generate 'nsim' simulations of 'nm2add' months
### beyond the observed time period:
nm2add <- 24
nsim <- 5
### 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)
## extend the 'stgrid' by replicating the last block 'nm2add' times
## (i.e., holding "popdensity" constant)
stgridext <- local({
    gLast <- subset(imdepi$stgrid, BLOCK == max(BLOCK))
    gAdd <- gLast[rep(1:nrow(gLast), nm2add),]; rownames(gAdd) <- NULL
    newstart <- seq(origT, by=30, length.out=nm2add)
    newstop <- c(newstart[-1], max(newstart) + 30)
```

```
    gAdd$start <- rep(newstart, each=nlevels(gAdd$tile))
    gAdd$stop <- rep(newstop, each=nlevels(gAdd$tile))
    rbind(imdepi$stgrid, gAdd, make.row.names = FALSE)[,-1]
})
## create an updated "epidataCS" with the time-extended 'stgrid'
imdepiext <- update(imdepi, stgrid = stgridext)
newT <- tail(imdepiext$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
plot(imdepiext, breaks=c(unique(imdepi$stgrid$start),origT),
     cumulative=list(maxat=330))
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),
         border=NA, density=0)  # no histogram
abline(v=origT, lty=2, lwd=2)


## End(Not run)
```

---

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", "imdepifit")

## simple baseline model
m0 <- update(imdepifit, epidemic=~1, siaf=NULL)

## 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", "imdepifit")

## add another epidemic covariate
## (but fix siaf-parameter so that this example runs quickly)
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 legacy wrapper for the polygon clipping engines implemented by packages **sf** and **polyclip**. Internally, both methods need to convert the input polygons to a class appropriate for the method, so are rather inefficient.

Usage

```
unionSpatialPolygons(SpP, method = c("sf", "polyclip"), ...)
```

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 call st_union in package **sf**. For method="polyclip", function polyclip from package **polyclip** is used. The old method="gpclib" is no longer available.
...	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

st_union in package **sf**, polyclip in package **polyclip**.

Examples

```
## Load districts of Germany
load(system.file("shapes", "districtsD.RData", package = "surveillance"))
plot(districtsD, border = "gray", asp = 1)

## Union these districts using either "sf" or "polyclip"
if (requireNamespace("sf"))  {
    stateD <- unionSpatialPolygons(districtsD, method = "sf")
    plot(stateD, add = TRUE, border = 2, lwd = 2)
}
if (requireNamespace("polyclip")) {
    stateD_pc <- unionSpatialPolygons(districtsD, method = "polyclip")
    plot(stateD_pc, add = TRUE, border = 1, lwd = 2, lty = 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	upper bound 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.
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. Note, however, that an update is forced within twinstim if a subset of the data is selected for model fitting or if a different qmatrix is supplied.
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.

Details

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 otherwise uses x "+-" runif(length(x), 0, amount).

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. Note that this implements a simplistic approach of tie-breaking where all events are assumed to be subject to the same amounts of censoring, and the default amounts may not be sensible choices.

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, control)
         return(control),verbose=TRUE,...)

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,
                          reset = FALSE), ...)

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 and the current control object and which returns the appropriate control object for region k.
verbose	Boolean, if TRUE then textual information about the process is given
...	currently ignored.

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.

---

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
```

Index

∗ **aplot**
addFormattedXAxis, 9
layout.labels, 166
twinSIR_intensityplot, 245
twinstim_iafplot, 273
twinstim_intensity, 276
∗ **chron**
formatDate, 107
isoWeekYear, 161
refvalIdxByDate, 201
∗ **classes**
epidata, 68
epidataCS, 81
sts-class, 214
stsBP-class, 224
stsNC-class, 225
∗ **classif**
algo.bayes, 12
algo.call, 14
algo.cdc, 15
algo.compare, 17
algo.cusum, 18
algo.farrington, 20
algo.glrnb, 25
algo.hmm, 29
algo.outbreakP, 32
algo.rki, 35
algo.rogerson, 37
boda, 48
earsC, 65
farringtonFlexible, 99
wrap.algo, 300
∗ **cluster**
stcd, 210
∗ **datagen**
discpoly, 63
hhh4_simulate, 135
sim.pointSource, 208
sim.seasonalNoise, 209
twinSIR_simulation, 252
twinstim_simEndemicEvents, 286
twinstim_simulation, 287
∗ **datasets**
abattoir, 8
campyDE, 54
deleval, 62
fluBYBW, 106
ha, 110
hagelloch, 110
hepatitisA, 113
husO104Hosp, 154
imdepi, 155
imdepifit, 158
influMen, 159
m1, 172
measles.weser, 175
measlesDE, 176
meningo.age, 177
MMRcoverageDE, 178
momo, 179
rotaBB, 203
salmAllOnset, 203
salmHospitalized, 204
salmNewport, 204
salmonella.agona, 205
shadar, 207
stsNewport, 227
∗ **distribution**
fanplot, 97
∗ **dplot**
bestCombination, 47
checkResidualProcess, 59
hhh4_validation, 145
layout.labels, 166
magic.dim, 173
pit, 192
twinSIR_intensityplot, 245
twinSIR_profile, 250

twinstim_intensity, 276
twinstim_profile, 283
untie, 298
∗ **dynamic**
animate, 41
epidata_animate, 73
epidataCS_animate, 89
sts_animate, 218
∗ **environment**
surveillance.options, 236
∗ **graphs**
poly2adjmat, 195
∗ **hplot**
animate, 41
epidata_animate, 73
epidata_plot, 77
epidataCS_animate, 89
epidataCS_plot, 93
fanplot, 97
hhh4_plot, 127
hhh4_simulate_plot, 138
intensityplot, 159
ks.plot.unif, 164
sts_animate, 218
sts_ggplot, 222
stsNClist_animate, 226
stsplot, 227
stsplot_space, 228
stsplot_time, 231
twinSIR_intensityplot, 245
twinstim_iafplot, 273
twinstim_intensity, 276
twinstim_plot, 282
∗ **htest**
calibrationTest, 53
checkResidualProcess, 59
hhh4_validation, 145
knox, 162
ks.plot.unif, 164
permutationTest, 190
stK, 212
twinSIR_methods, 248
twinSIR_profile, 250
twinstim_epitest, 265
twinstim_methods, 279
twinstim_profile, 283
∗ **iteration**
clapply, 60
plapply, 193
∗ **list**
clapply, 60
plapply, 193
∗ **manip**
epidata, 68
epidata_intersperse, 76
epidataCS, 81
epidataCS_aggregate, 87
epidataCS_permute, 92
epidataCS_update, 96
intersectPolyCircle, 160
tidy.sts, 237
untie, 298
∗ **math**
primeFactors, 197
∗ **methods**
aggregate-methods, 11
epidata_plot, 77
epidata_summary, 79
epidataCS_aggregate, 87
epidataCS_plot, 93
epidataCS_update, 96
hhh4_methods, 124
hhh4_predict, 133
hhh4_update, 143
R0, 198
residualsCT, 202
sts-class, 214
sts_conversion, 220
stsplot, 227
stsSlot-generics, 234
stsXtrct, 234
twinSIR_intensityplot, 245
twinSIR_methods, 248
twinSIR_profile, 250
twinstim_intensity, 276
twinstim_methods, 279
twinstim_profile, 283
twinstim_step, 293
twinstim_update, 296
∗ **misc**
algo.quality, 34
∗ **models**
arlCusum, 42
backprojNP, 43
coeflist, 61
find.kh, 103

findH, 104
findK, 105
glm_epidataCS, 108
hhh4_predict, 133
hhh4_update, 143
hhh4_W, 150
linelist2sts, 167
nowcast, 182
twinSIR, 240
twinSIR_simulation, 252
twinstim, 257
twinstim_iaf, 268
twinstim_simEndemicEvents, 286
twinstim_simulation, 287
twinstim_step, 293
twinstim_update, 296

∗ **optimize**
backprojNP, 43
linelist2sts, 167
twinSIR, 240
twinSIR_profile, 250
twinstim, 257
twinstim_profile, 283

∗ **package**
surveillance-package, 6

∗ **print**
algo.summary, 39
formatPval, 108
hhh4_methods, 124
toLatex.sts, 238
twinSIR_methods, 248
twinstim_methods, 279

∗ **regression**
algo.farrington.assign.weights, 23
algo.farrington.fitGLM, 23
algo.farrington.threshold, 25
anscombe.residuals, 41
categoricalCUSUM, 56
hhh4, 114
hhh4_formula, 123
LRCUSUM.runlength, 169
pairedbinCUSUM, 187

∗ **spatial**
animate, 41
discpoly, 63
epidata, 68
epidata_animate, 73
epidata_intersperse, 76
epidata_plot, 77
epidataCS, 81
epidataCS_aggregate, 87
epidataCS_animate, 89
epidataCS_plot, 93
hhh4_W, 150
intersectPolyCircle, 160
multiplicity.Spatial, 180
nbOrder, 181
poly2adjmat, 195
polyAtBorder, 196
sts_animate, 218
stsplot, 227
stsplot_space, 228
unionSpatialPolygons, 297
zetaweights, 301

∗ **ts**
hhh4, 114
hhh4_validation, 145
sts-class, 214
sts_conversion, 220
sts_ggplot, 222
stsplot, 227
stsplot_time, 231
tidy.sts, 237

∗ **univar**
hhh4_simulate_scores, 141
hhh4_validation, 145
R0, 198
scores, 205

∗ **utilities**
all.equal, 40
coeflist, 61
disProg2sts, 64
epidataCS_update, 96
hhh4_W, 150
hhh4_W_utils, 153
magic.dim, 173
multiplicity.Spatial, 180
nbOrder, 181
twinstim_iaf, 268
twinstim_siaf, 284
twinstim_tiaf, 295
untie, 298
zetaweights, 301

.GlobalEnv, *11*
.Random.seed, *135*, *162*, *191*, *194*, *274*, *290*
[,sts,ANY,ANY,ANY-method (stsXtrct), 234

[,sts-method (stsXtrct), 234
[.data.frame, *71*
[.epidata (epidata), 68
[.epidataCS (epidataCS), 81

abattoir, 8
abline, *130*
add1, *293, 294*
add1.default, *294*
add1.twinstim (twinstim_step), 293
addFormattedXAxis, 9, *129*, *139*, *232*
addSeason2formula, 10, *116*, *125*, *143*, *174*
aggregate,sts-method (aggregate-methods), 11
aggregate-methods, 11
aggregate.hhh4sims (hhh4_simulate_plot), 138
aggregate.hhh4simslist (hhh4_simulate_plot), 138
aggregate.sts, *217*
aggregate.sts (aggregate-methods), 11
aggregate.ts, *11*
aggregated, *228*
AIC, *126*, *244*, *250*, *279*
AIC.twinSIR (twinSIR_methods), 248
alarms, *49*, *51*
alarms (stsSlot-generics), 234
alarms,sts-method (sts-class), 214
alarms<- (stsSlot-generics), 234
alarms<-,sts-method (sts-class), 214
algo.bayes, 12, *15, 16*, *37*
algo.bayes1 (algo.bayes), 12
algo.bayes2 (algo.bayes), 12
algo.bayes3 (algo.bayes), 12
algo.bayesLatestTimepoint, *16*, *37*
algo.bayesLatestTimepoint (algo.bayes), 12
algo.call, 14, *14*
algo.cdc, 15
algo.cdcLatestTimepoint (algo.cdc), 15
algo.compare, 17, *35*, *39*
algo.cusum, 18, *301*
algo.farrington, *15*, 20, *24*, *301*
algo.farrington.assign.weights, 23
algo.farrington.fitGLM, *21, 22*, 23, *102*
algo.farrington.threshold, *22*, 25, *102*
algo.glrnb, 25, *301*
algo.glrpois, *301*
algo.glrpois (algo.glrnb), 25
algo.hmm, 29
algo.outbreakP, 32, *301*
algo.quality, *17*, 34, *39*
algo.rki, *14, 15*, 35, *301*
algo.rki1 (algo.rki), 35
algo.rki2 (algo.rki), 35
algo.rki3 (algo.rki), 35
algo.rkiLatestTimepoint, *14*, *16*
algo.rkiLatestTimepoint (algo.rki), 35
algo.rogerson, 37
algo.summary, 39
all.equal, 40, *40*
ani.options, *75*
animate, 41, *70*, *228*
animate.epidata, *41*, *72*, *79*, *255*
animate.epidata (epidata_animate), 73
animate.epidataCS, *41*, *85*, *95*, *291*
animate.epidataCS (epidataCS_animate), 89
animate.sts, *41*, *217*, *230*
animate.sts (sts_animate), 218
animate.summary.epidata (epidata_animate), 73
animate_nowcasts (stsNClist_animate), 226
anscombe.residuals, *23, 24*, 41
arlCusum, 42
as.data.frame,sts-method (tidy.sts), 237
as.data.frame.sts, *217*
as.data.frame.sts (tidy.sts), 237
as.epidata, *81*, *88*, *111, 112*, *242*, *244*, *253*
as.epidata (epidata), 68
as.epidata.epidataCS, *77*, *85*
as.epidata.epidataCS (epidataCS_aggregate), 87
as.epidataCS, *289, 290*, *299*
as.epidataCS (epidataCS), 81
as.factor, *139*
as.hhh4simslist (hhh4_simulate_plot), 138
as.matrix, *71*
as.stepfun, *85*
as.stepfun.epidataCS (epidataCS), 81
as.ts, *217*
as.ts.sts (sts_conversion), 220
as.xts.sts, *217*
as.xts.sts (sts_conversion), 220
as.xts.ts, *220*

at2ndChange (addFormattedXAxis), 9
atChange (addFormattedXAxis), 9
atMedian (addFormattedXAxis), 9
autoplot, *228*, *233*
autoplot.sts, *217*
autoplot.sts (sts_ggplot), 222
axis, *9*
axisTicks, *130*

backprojNP, 43
bayes (wrap.algo), 300
bestCombination, 47, *173*
BIC, *126*
boda, 48
bodaDelay, 50
boxplot, *131*

calc.outbreakP.statistic (algo.outbreakP), 32
calibrationTest, 53, *145*, *148*
calibrationTest.default, *147*
calibrationTest.hhh4 (hhh4_validation), 145
calibrationTest.oneStepAhead (hhh4_validation), 145
campyDE, 54
catcusum.LLRcompute (categoricalCUSUM), 56
categoricalCUSUM, *6*, *8*, 56, *171*, *189*
checkResidualProcess, 59, *165*, *203*
citation, *7*
clapply, 60
class, *77*, *80*
clusterSetRNGStream, *194*
coef, *126*, *281*
coef.epitest (twinstim_epitest), 265
coef.hhh4, *11*
coef.hhh4 (hhh4_methods), 124
coeflist, 61, *281*, *290*
coeflist.hhh4 (hhh4_methods), 124
coeflist.twinstim (twinstim_methods), 279
coefW, *152*
coefW (hhh4_W_utils), 153
coerce,epidataCS,SpatialPointsDataFrame-method (epidataCS), 81
coerce,sts,stsBP-method (stsBP-class), 224
coerce,sts,stsNC-method (stsNC-class), 225
coerce,sts,ts-method (sts_conversion), 220
coerce,ts,sts-method (sts_conversion), 220
colorRampPalette, *97*, *139*
confint, *126*, *279*, *281*
confint.hhh4 (hhh4_methods), 124
confint.oneStepAhead (hhh4_validation), 145
control (stsSlot-generics), 234
control,sts-method (sts-class), 214
control<- (stsSlot-generics), 234
control<-,sts-method (sts-class), 214
coordinates, *181*
cox, *241*, *254*
cox (twinSIR_cox), 245
cusum (wrap.algo), 300
cycle, *216*, *238*

data.frame, *69*, *71*, *82*
Date, *94*, *107*, *161*, *215*, *220*, *237*, *238*
delayCDF (stsNC-class), 225
delayCDF,stsNC-method (stsNC-class), 225
deleval, 62
dev.interactive, *74*, *90*, *219*
dev.print, *75*
dim, *215*
dim,sts-method (sts-class), 214
dimnames, *216*
dimnames,sts-method (sts-class), 214
disc, *63*
discpoly, 63, *83*, *160*
disProg2sts, 64, *110*, *214*
dist, *71*, *162*
dnbinom, *53*, *206*
drop1, *293*, *294*
drop1.default, *294*
drop1.twinstim (twinstim_step), 293
dss, *147*
dss (scores), 205

earsC, 65
ecdf, *164*
environment, *260*
epidata, 68, *77*, *80*, *85*, *87*, *88*, *112*, *240*, *245*, *252*, *255*
epidata_animate, 73

epidata_intersperse, 76
epidata_plot, 77
epidata_summary, 79
epidataCS, 81, *87, 88*, *92, 93*, *96, 97*, *109*, *155, 156*, *258*, *266*, *269*, *277*, *288*
epidataCS2sts, *85*
epidataCS2sts *(epidataCS_aggregate)*, 87
epidataCS_aggregate, 87
epidataCS_animate, 89
epidataCS_permute, 92
epidataCS_plot, 93
epidataCS_update, 96
epidataCSplot_space *(epidataCS_plot)*, 93
epidataCSplot_time *(epidataCS_plot)*, 93
epitest, *92*, *164*, *199*, *214*, *259*
epitest *(twinstim_epitest)*, 265
epoch *(stsSlot-generics)*, 234
epoch,sts-method *(sts-class)*, 214
epoch<- *(stsSlot-generics)*, 234
epoch<-,sts-method *(sts-class)*, 214
epochInYear *(sts-class)*, 214
epochInYear,sts-method *(sts-class)*, 214
extractAIC, *244*, *250*
extractAIC.twinSIR, *249*
extractAIC.twinSIR *(twinSIR_methods)*, 248

facet_wrap, *223*
factor, *70*, *131*
fan, *97, 98*, *139*
fanplot, 97, *147*
farrington *(algo.farrington)*, 20
farringtonFlexible, *22*, 99
fe, *10*, *116*, *119*
fe *(hhh4_formula)*, 123
find.kh, 103
findH, *37*, 104
findK, 105
fixef *(ranef)*, 201
fixef.hhh4, *201*
fixef.hhh4 *(hhh4_methods)*, 124
fluBYBW, 106
format, *108*
format.Date, *107*
format.pval, *108*, *280*
formatDate, 107, *168, 169*
formatPval, 108
formula, *10, 11*, *126*, *240*, *252*
formula.hhh4 *(hhh4_methods)*, 124
frequency, *238*
frequency,sts-method *(sts-class)*, 214

geom_col, *223*
geom_line, *223*
getMaxEV *(hhh4_plot)*, 127
getMaxEV_season *(hhh4_plot)*, 127
getNEweights, *152*
getNEweights *(hhh4_W_utils)*, 153
getSourceDists, *269*
getSourceDists *(epidataCS)*, 81
glm, *41*, *108, 109*
glm.nb, *26*, *117*
glm_epidataCS, 108
glrnb, *26*
glrnb *(wrap.algo)*, 300
glrpois *(wrap.algo)*, 300
grid, *98*
grid.arrange, *131*
grid.text, *230*

h1_nrwrp *(m1)*, 172
ha, 110
hagelloch, *72*, 110, *181*
hcl.colors, *230*
head.epidataCS *(epidataCS)*, 81
hepatitisA, 113
hhh4, *6*, *10, 11*, *87, 88*, 114, *123, 124*, *126–128*, *133*, *135*, *138*, *141*, *143–146*, *150, 151*, *153*, *174*, *201*, *301*
hhh4_formula, 123
hhh4_methods, 124
hhh4_plot, 127
hhh4_predict, 133
hhh4_simulate, 135
hhh4_simulate_plot, 138
hhh4_simulate_scores, 141
hhh4_update, 143
hhh4_validation, 145
hhh4_W, 150
hhh4_W_utils, 153
hist, *93–95*, *192*
hist.Date, *93, 94*
husO104Hosp, 154
hValues, *38*
hValues *(findH)*, 104

iafplot, *282, 283*

iafplot (twinstim_iafplot), 273
imdepi, 155, *158*
imdepifit, 158
influMen, 159
integrate, *259*
intensity.twinstim
(twinstim_intensity), 276
intensityplot, 159, *245*, *260*, *276*, *290*
intensityplot.simEpidata, *255*
intensityplot.simEpidata
(twinSIR_intensityplot), 245
intensityplot.simEpidataCS
(twinstim_intensity), 276
intensityplot.twinSIR, *160*
intensityplot.twinSIR
(twinSIR_intensityplot), 245
intensityplot.twinstim, *160*, *282*, *283*, *296*
intensityplot.twinstim
(twinstim_intensity), 276
interactive, *194*, *219*
intersect.owin, *83*
intersectPolyCircle, 160
intersperse (epidata_intersperse), 76
is.projected, *166*
isoWeekYear, 161, *215*

jitter, *298–300*

k1 (m1), 172
knots, *253*
knox, 162, *214*, *265*, *267*
ks.plot.unif, *59*, *60*, 164
ks.test, *164*, *165*

lapply, *60*, *194*
layout.labels, *131*, 166, *229*
layout.scale.bar, *166*
layout.scalebar (layout.labels), 166
legend, *74*, *78*, *90*, *94*, *95*, *129*, *139*, *232*, *275*
levelplot, *130*, *131*, *230*
linelist2sts, *107*, 167, *184*
lines, *97*, *98*, *139*, *146*
list, *216*
lm, *241*, *254*
log_breaks, *130*
logLik, *126*, *244*, *279*
logLik.hhh4 (hhh4_methods), 124
logLik.twinSIR (twinSIR_methods), 248
logLik.twinstim (twinstim_methods), 279
logs (scores), 205
LRCUSUM.runlength, *57*, 169

m1, 172
m2 (m1), 172
m3 (m1), 172
m4 (m1), 172
m5 (m1), 172
magic.dim, *47*, 173
makeCluster, *194*
makeControl, *116*, 174
mapply, *174*, *192*
marks, *85*, 174
marks.epidataCS, *174*
marks.epidataCS (epidataCS), 81
matlines, *139*
matplot, *130*, *232*, *246*
Matrix, *71*
matrix, *69*
mclapply, *146*, *193*, *194*, *260*
meanHHH, *134*
measles.weser, 175
measlesDE, 176, *179*
measlesWeserEms (measles.weser), 175
meningo.age, 177
missing, *232*
MMRcoverageDE, *177*, 178
modifyList, *143*, *230*, *296*
momo, 179
msm, *29–31*
multinomialTS (stsSlot-generics), 234
multinomialTS,sts-method (sts-class), 214
multinomialTS<- (stsSlot-generics), 234
multinomialTS<-,sts-method (sts-class), 214
multiplicity, 180, *181*
multiplicity.Spatial, 180, *180*

n1 (m1), 172
n2 (m1), 172
n2mfrow, *173*
na.pass, *258*
NaN, *259*
nb2mat, *195*
nblag, *182*
nbOrder, *131*, *151*, *152*, 181, *301*
negative.binomial, *26*

neighbourhood (stsSlot-generics), 234
neighbourhood,sts-method (sts-class), 214
neighbourhood<- (stsSlot-generics), 234
neighbourhood<-,sts-method (sts-class), 214
nlm, *116*
nlme::ranef, *201*
nlminb, *117*, *260*, *271*
nobs, *126*, *279*
nobs.epidataCS (epidataCS), 81
nobs.hhh4 (hhh4_methods), 124
nobs.twinstim (twinstim_methods), 279
nowcast, 182

observed (stsSlot-generics), 234
observed,sts-method (sts-class), 214
observed<- (stsSlot-generics), 234
observed<-,sts-method (sts-class), 214
on.exit, *194*
oneStepAhead, *143*
oneStepAhead (hhh4_validation), 145
optim, *103*, *116*, *240*, *241*, *251*, *259*, *283*
options, *236*
outbreakP (wrap.algo), 300
owin, *63*, *160*

pairedbinCUSUM, *6*, *62*, 187
panel.text, *166*
par, *59*, *74*, *78*, *94*, *129*, *139*, *213*, *232*, *236*
parLapply, *193*, *194*
permutationTest, 190
permute.epidataCS, *266*, *267*
permute.epidataCS (epidataCS_permute), 92
persp, *213*
pit, *145*, *148*, 192
pit.default, *147*
pit.hhh4 (hhh4_validation), 145
pit.oneStepAhead (hhh4_validation), 145
plapply, *162*, 193, *213*, *266*
plot, *72*, *77*, *127*, *231*, *260*
plot,sts,missing-method (stsplot), 227
plot,stsNC,missing-method (stsplot), 227
plot.default, *98*, *213*, *274*
plot.epidata, *75*, *255*
plot.epidata (epidata_plot), 77
plot.epidataCS, *85*, *91*, *291*
plot.epidataCS (epidataCS_plot), 93
plot.epitest (twinstim_epitest), 265
plot.function, *79*
plot.hhh4 (hhh4_plot), 127
plot.hhh4sims, *98*, *136*
plot.hhh4sims (hhh4_simulate_plot), 138
plot.hhh4simslist (hhh4_simulate_plot), 138
plot.histogram, *192*
plot.knox (knox), 162
plot.oneStepAhead, *98*
plot.oneStepAhead (hhh4_validation), 145
plot.profile.twinSIR (twinSIR_profile), 250
plot.stepfun, *79*, *274*
plot.stKtest (stK), 212
plot.sts, *217*
plot.sts (stsplot), 227
plot.summary.epidata (epidata_plot), 77
plot.survRes, *233*
plot.twinSIR, *244*
plot.twinSIR (twinSIR_intensityplot), 245
plot.twinstim, *158*, *275*, *278*
plot.twinstim (twinstim_plot), 282
plotHHH4_fitted (hhh4_plot), 127
plotHHH4_fitted1, *9*, *10*
plotHHH4_fitted1 (hhh4_plot), 127
plotHHH4_maps (hhh4_plot), 127
plotHHH4_maxEV (hhh4_plot), 127
plotHHH4_neweights (hhh4_plot), 127
plotHHH4_ri (hhh4_plot), 127
plotHHH4_season (hhh4_plot), 127
plotHHH4sims_fan (hhh4_simulate_plot), 138
plotHHH4sims_size (hhh4_simulate_plot), 138
plotHHH4sims_time (hhh4_simulate_plot), 138
points, *74*, *95*
poly2adjmat, *87*, 195
poly2nb, *195*
polyAtBorder, 196
polyclip, *297*, *298*
polyCub, *269*, *285*
polyCub.iso, *259*, *270*
polyCub.midpoint, *259*, *269*
polyCub.SV, *259*
Polygon, *63*

population (stsSlot-generics), 234
population,sts-method (sts-class), 214
population<- (stsSlot-generics), 234
population<-,sts-method (sts-class), 214
predefined siaf's, *288*
predict.hhh4 (hhh4_predict), 133
predint (stsNC-class), 225
predint,stsNC-method (stsNC-class), 225
pretty, *229*
primeFactors, *173*, 197
print, *80*, *279*
print.algoQV (algo.quality), 34
print.data.frame, *71*, *84*
print.epidata (epidata), 68
print.epidataCS (epidataCS), 81
print.hhh4 (hhh4_methods), 124
print.Latex, *281*
print.summary.epidata (epidata_summary), 79
print.summary.epidataCS (epidataCS), 81
print.summary.twinSIR (twinSIR_methods), 248
print.summary.twinstim (twinstim_methods), 279
print.table, *84*
print.twinSIR (twinSIR_methods), 248
print.twinstim (twinstim_methods), 279
print.xtable, *239*, *281*
printCoefmat, *249*, *280*, *281*
proc.time, *118*, *262*
profile.twinSIR, *244*
profile.twinSIR (twinSIR_profile), 250
profile.twinstim (twinstim_profile), 283
proj4string, *83*, *166*

q1_nrwh (m1), 172
q2 (m1), 172
quantile.oneStepAhead (hhh4_validation), 145

R0, 198, *260*, *290*
rainbow, *94*
ranef, 201
ranef.hhh4, *201*
ranef.hhh4 (hhh4_methods), 124
refvalIdxByDate, 201
reportingTriangle (stsNC-class), 225
reportingTriangle,stsNC-method (stsNC-class), 225
reset.surveillance.options (surveillance.options), 236
reshape, *237*
residuals, *59*, *60*, *126*, *260*, *290*
residuals.glm, *126*
residuals.hhh4 (hhh4_methods), 124
residuals.simEpidataCS (residualsCT), 202
residuals.twinSIR (residualsCT), 202
residuals.twinstim, *260*, *261*
residuals.twinstim (residualsCT), 202
residualsCT, 202
ri, *116*, *119*
ri (hhh4_formula), 123
rki (wrap.algo), 300
rnbinom, *135*, *221*
RNGkind, *194*
rotaBB, 203
rps (scores), 205
rug, *78*, *246*, *273*, *277*
run.jags, *185*, *186*

s1 (m1), 172
s2 (m1), 172
s3 (m1), 172
salmAllOnset, 203
salmHospitalized, 204
salmNewport, 204
salmonella.agona, 205
saveHTML, *89–91*, *218*
score (stsNC-class), 225
score,stsNC-method (stsNC-class), 225
scores, *126*, *145*, *191*, 205
scores.hhh4 (hhh4_validation), 145
scores.hhh4sims, *136*
scores.hhh4sims (hhh4_simulate_scores), 141
scores.hhh4simslist (hhh4_simulate_scores), 141
scores.oneStepAhead (hhh4_validation), 145
seq.Date, *168*, *169*, *184*
ses (scores), 205
set.seed, *49*, *135*, *194*, *213*, *254*, *290*
sf, *215*
shadar, 207
siaf, *258*, *268*, *270–272*, *296*
siaf (twinstim_siaf), 284
siaf.constant (twinstim_iaf), 268

siaf.exponential, *258*
siaf.exponential (twinstim_iaf), 268
siaf.gaussian, *258, 259*, *284*, *286*
siaf.gaussian (twinstim_iaf), 268
siaf.powerlaw, *258, 259*
siaf.powerlaw (twinstim_iaf), 268
siaf.powerlaw1 (twinstim_iaf), 268
siaf.powerlawL (twinstim_iaf), 268
siaf.step, *258*, *260*, *274*
siaf.step (twinstim_iaf), 268
siaf.student (twinstim_iaf), 268
sim.pointSource, 208, *210*
sim.seasonalNoise, *208*, 209
simEndemicEvents, *266*, *291*
simEndemicEvents (twinstim_simEndemicEvents), 286
simEpidata, *72*, *245, 246*
simEpidata (twinSIR_simulation), 252
simEpidataCS, *202*, *276*
simEpidataCS (twinstim_simulation), 287
simpleR0, *266, 267*
simpleR0 (R0), 198
simplify.owin, *85*, *291*
simplify2array, *206*
simulate, *135*, *138*, *141*, *241*, *252*, *260*, *287*, *290*
simulate.hhh4, *138*, *229*
simulate.hhh4 (hhh4_simulate), 135
simulate.twinSIR, *244*, *247*
simulate.twinSIR (twinSIR_simulation), 252
simulate.twinstim, *158*, *262*, *286, 287*
simulate.twinstim (twinstim_simulation), 287
sp.points, *277*
sp.polygons, *131*, *230*, *277*
Spatial, *166*, *180, 181*
SpatialPixels, *277*
SpatialPoints, *181*
SpatialPointsDataFrame, *82*, *84*, *155*, *287*, *289*, *291*
SpatialPolygons, *83, 84*, *88*, *91*, *93, 94*, *131*, *156*, *195, 196*, *215, 216*, *229*, *266*, *277*, *286*, *289*, *297*, *299*
SpatialPolygonsDataFrame, *94*, *106*, *175*, *215, 216*, *277*
spatstat.geom::marks, *174*
spatstat.geom::multiplicity, *180*
split, *61*
spplot, *93–95*, *128*, *131*, *166*, *228–230*, *277*
sprintf, *280*
sqrt_trans, *139*, *229*
st_union, *297, 298*
start,sts-method (sts-class), 214
stateplot (epidata_plot), 77
stcd, 210
stdiagn, *213*
step, *293, 294*
stepComponent (twinstim_step), 293
stepfun, *253*
stK, 212
stkhat, *213*
stKtest, *164*
stKtest (stK), 212
stmctest, *213*
storage.mode, *85*
strftime, *9, 10*, *107*, *161*
stripplot, *131*
strptime, *168, 169*, *236*
sts, *6*, *9*, *11, 12*, *21, 22*, *26*, *44*, *62*, *85*, *87, 88*, *99*, *106*, *115*, *127*, *129*, *135, 136*, *168*, *175, 176*, *179*, *204*, *218*, *220, 221*, *223–225*, *227–229*, *231*, *234, 235*, *237–239*, *300, 301*
sts (sts-class), 214
sts-class, 214
sts2disProg, *26*
sts2disProg (disProg2sts), 64
sts_animate, 218
sts_conversion, 220
sts_creation, 221
sts_ggplot, 222
sts_observation, 224
stsBP, *44, 45*
stsBP-class, 224
stsecal, *213*
stsNC, *186*, *226*
stsNC-class, 225
stsNClist_animate, 226
stsNewport, 227
stsplot, *219*, 227, *230*, *233*
stsplot_alarm (stsplot_time), 231
stsplot_space, *218*, 228, *228*
stsplot_time, *129*, *222, 223*, *228*, 231
stsplot_time1, *9, 10*, *228*, *236*

stsplot_time1 (stsplot_time), 231
stsSlot-generics, 234
stsXtrct, 234
subset.data.frame, *84*, *93*
subset.epidataCS (epidataCS), 81
summary, *72*, *80*, *260*, *279*
summary.epidata, *71*, *75*, *79*
summary.epidata (epidata_summary), 79
summary.epidataCS (epidataCS), 81
summary.hhh4, *132*
summary.hhh4 (hhh4_methods), 124
summary.twinSIR (twinSIR_methods), 248
summary.twinstim, *158*
summary.twinstim (twinstim_methods), 279
surveillance (surveillance-package), 6
surveillance-package, 6
surveillance.options, 236
Sys.sleep, *74*, *90*

t.test, *191*
tail.epidataCS (epidataCS), 81
terms, *116*, *245*
text, *166*
tiaf, *258*, *268*, *271*, *272*, *286*
tiaf (twinstim_tiaf), 295
tiaf.constant, *288*
tiaf.constant (twinstim_iaf), 268
tiaf.exponential, *258*, *288*, *295*, *296*
tiaf.exponential (twinstim_iaf), 268
tiaf.step, *258*, *274*
tiaf.step (twinstim_iaf), 268
tidy.sts, *222*, 237
title, *74*
toLatex, *260*, *279*
toLatex,sts-method (toLatex.sts), 238
toLatex.knox (knox), 162
toLatex.sts, *217*, 238
toLatex.summary.twinstim (twinstim_methods), 279
trans, *229*
trellis, *219*, *230*
trellis.object, *94*, *95*, *131*, *278*
truehist, *162*, *191*, *213*, *266*, *267*
ts, *215*, *220*
twinSIR, *6*, *59*, *68*, *70*, *72*, *87*, *88*, *111*, *112*, *202*, 240, *245–247*, *252*, *255*, *262*
twinSIR_cox, 245
twinSIR_intensityplot, 245
twinSIR_methods, 248
twinSIR_profile, 250
twinSIR_simulation, 252
twinstim, *6*, *59*, *81*, *83–85*, *108*, *155*, *156*, *158*, *164*, *198*, *199*, *202*, *214*, 257, *265*, *272*, *276*, *281*, *282*, *284*, *286–288*, *290*, *291*, *293–296*
twinstim_epitest, 265
twinstim_iaf, 268
twinstim_iafplot, 273
twinstim_intensity, 276
twinstim_methods, 279
twinstim_plot, 282
twinstim_profile, 283
twinstim_siaf, 284
twinstim_simEndemicEvents, 286
twinstim_simulation, 287
twinstim_step, 293
twinstim_tiaf, 295
twinstim_update, 296
txtProgressBar, *76*, *194*, *219*

unionSpatialPolygons, *196*, *289*, 297
untie, *82*, *155*, *258*, 298
update, *85*, *96*, *112*, *127*, *286*
update.default, *297*
update.epidata (epidata), 68
update.epidataCS (epidataCS_update), 96
update.formula, *296*
update.hhh4 (hhh4_update), 143
update.twinstim (twinstim_update), 296
upperbound, *49*, *51*
upperbound (stsSlot-generics), 234
upperbound,sts-method (sts-class), 214
upperbound<- (stsSlot-generics), 234
upperbound<-,sts-method (sts-class), 214

vcov, *126*, *244*, *279*
vcov.hhh4 (hhh4_methods), 124
vcov.twinSIR (twinSIR_methods), 248
vcov.twinstim (twinstim_methods), 279

W_np (hhh4_W), 150
W_powerlaw, *115*, *117*, *153*, *301*, *302*
W_powerlaw (hhh4_W), 150
wrap.algo, *6*, *21*, 300

xtable, *260*, *279–281*
xtable.algoQV (algo.quality), 34
xtable.summary.twinstim (twinstim_methods), 279

xtable.twinstim *(twinstim_methods)*, 279
xts, *220*
xylist, *213*, *297*

year *(sts-class)*, 214
year,sts-method *(sts-class)*, 214

zetaweights, 301