

# Package ‘surveillance’

January 15, 2012

**Title** Modeling and monitoring discrete response time series

**Version** 1.2-1

**Date** 2010-05-25

**Author** M. Höhle with contributions from T. Correa, M. Hofmann, C. Lang, M. Paul, A. Riebler, S. Steiner, M. Virtanen and V. Wimmer

**Depends** methods,utils,xtable,spc,sp,maptools,vcd,msm,Matrix

**Suggests** RUnit,digest,coda,gamlss,splancs

**Description** A package implementing statistical methods for the modeling and change-point detection in time series of counts, proportions and categorical data. Focus is on outbreak detection in count data time series originating from public health surveillance of infectious diseases, but applications could just as well originate from environmetrics, reliability engineering, econometrics or social sciences. Currently the package contains implementations typical outbreak detection procedures such as Stroup et. al (1989), Farrington et al.(1996), Rossi et al. (1999), Rogerson and Yamada (2001), a Bayesian approach, negative binomial CUSUM methods and a detector based on generalized likelihood ratios. Furthermore, inference methods for the retrospective infectious disease model in Held et al. (2005), Held et al. (2006) and Paul et al. (2008) are provided. A novel CUSUM approach combining logistic and multinomial logistic modelling is also included. 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.

**Maintainer** Michael Höhle <hoehle@stat.uni-muenchen.de>

**License** GPL-2

**URL** <http://surveillance.r-forge.r-project.org/>

**ZipData** no

**Encoding** latin1

**Repository** CRAN

**Date/Publication** 2010-06-10 06:27:03

**R topics documented:**

surveillance-package	3
abattoir	5
aggregate-methods	6
aggregate.disProg	6
algo.bayes	7
algo.call	9
algo.cdc	10
algo.compare	12
algo.cusum	13
algo.farrington	15
algo.farrington.assign.weights	17
algo.farrington.fitGLM	18
algo.farrington.threshold	19
algo.glrnb	19
algo.glrpois	22
algo.hhh	24
algo.hhh.grid	27
algo.hmm	30
algo.outbreakP	32
algo.quality	34
algo.rki	36
algo.rogerson	37
algo.summary	39
algo.twins	41
anscombe.residuals	43
arlCusum	43
bestCombination	44
categoricalCUSUM	45
CIdata	48
compMatrix.writeTable	48
correct53to52	49
create.disProg	50
create.grid	51
deval	53
display-methods	54
disProg2sts	56
enlargeData	57
estimateGLRNbHook	58
estimateGLRPoisHook	59
find.kh	60
findH	61
findK	62
ha	63
hepatitisA	64
influMen	64
isoWeekYear	65

loglikelihood . . . . .	66
LRCUSUM.runlength . . . . .	66
m1 . . . . .	69
magic.dim . . . . .	70
make.design . . . . .	71
makePlot . . . . .	72
meanResponse . . . . .	73
measles.weser . . . . .	75
meningo.age . . . . .	75
momo . . . . .	76
observed-methods . . . . .	77
obsinyear-methods . . . . .	77
pairedbinCUSUM . . . . .	78
plot.atwins . . . . .	81
plot.disProg . . . . .	82
plot.survRes . . . . .	84
predict.ah . . . . .	86
primeFactors . . . . .	87
print.algoQV . . . . .	87
readData . . . . .	88
refvalIdxByDate . . . . .	89
residuals.ah . . . . .	90
salmonella.agona . . . . .	90
shadar . . . . .	91
sim.pointSource . . . . .	92
sim.seasonalNoise . . . . .	93
simHHH . . . . .	94
stcd . . . . .	96
sts-class . . . . .	98
sumNeighbours . . . . .	100
test . . . . .	100
testSim . . . . .	101
toFileDisProg . . . . .	102
wrap.algo . . . . .	103
xtable.algoQV . . . . .	104
year-methods . . . . .	105
[-methods . . . . .	105

**Index****107**

## Description

A package implementing statistical methods for the modeling and change-point detection in time series of counts, proportions and categorical data. Focus is on outbreak detection in count data time series originating from public health surveillance of infectious diseases, but applications could just as well originate from environmetrics, reliability engineering, econometrics or social sciences.

## Details

Package: surveillance  
Type: Package  
Version: 1.1-0  
Date: 2009-10-14  
License: GPL version 2 (<http://www.gnu.org/licenses/gpl.html>)

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

The fundamental data structure of the package is an S4 class `sts` wrapping observations, monitoring results and date handling for multivariate time series. Currently the package contains implementations typical outbreak detection procedures such as Stroup et al. (1989), Farrington et al., (1996), Rossi et al. (1999), Rogerson and Yamada (2001), a Bayesian approach (Höhle, 2007), negative binomial CUSUM methods (Höhle and Mazick, 2009), and a detector based on generalized likelihood ratios (Höhle and Paul, 2008). However, also CUSUMs for the prospective change-point detection in binomial, beta-binomial and multinomial time series is covered based on generalized linear modelling. This includes e.g. paired binary CUSUM described by Steiner et al. (1999) or paired comparison Bradley-Terry modelling described in Höhle (2010). The package contains several real-world datasets, the ability to simulate outbreak data, visualize the results of the monitoring in temporal, spatial or spatio-temporal fashion.

Furthermore, inference methods for the retrospective infectious disease model in Held et al. (2005) and Paul et al. (2008) handling multivariate time series of counts. Finally, the fully Bayesian approach for univariate time series of counts from Held et al. (2006) is also implemented.

## Author(s)

Author: M. Höhle with contributions from T. Correa, M. Hofmann, C. Lang, M. Paul, A. Riebler, S. Steiner and V. Wimmer

Maintainer: Michael Höhle <<hoehle@stat.uni-muenchen.de>>

## References

surveillance: An R package for the surveillance of infectious diseases (2007), M. Höhle, *Computational Statistics*, 22(4), pp. 571—582.

## Examples

```
#Code from an early survey article about the package: Hoehle (2007)
#available from http://surveillance.r-forge.r-project.org/
## Not run: demo(cost)
#Code from a more recent book chapter about using the package for the
#monitoring of Danish mortality data (Hoehle, 2009).
## Not run: demo(biosurvbook)
```

---

abattoir

*Abattoir Data*

---

## Description

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

## Usage

```
data(abattoir)
```

## Details

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

## References

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

## See Also

[categoricalCUSUM](#)

## Examples

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

---

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

---

### Description

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

Note: The function is not 100% consistent with what the generic function [aggregate](#) does.

### Details

Warning: In case the aggregation is by unit the upperbound slot is set to NA. Furthermore the MAP object is left as is, but the object cannot be plotted anymore.

### Methods

`x = "sts", by="time", nfreq="all",...` `x` an object of class `sts`  
`by` a string being either "time" or "unit"  
`nfreq` new sampling frequency if `by=="time"`. If `nfreq=="all"` then all time instances are summed.  
`...` not used  
 returns an object of class `sts`

### See Also

[aggregate](#)

### Examples

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

---

aggregate.disProg      *Aggregate the observed counts*

---

### Description

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

**Usage**

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

**Arguments**

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

**Value**

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

**Examples**

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

---

 algo.bayes

*The Bayes System*


---

**Description**

Evaluation of timepoints with the Bayes subsystem 1,2 or 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.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. 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 for calculating an upper limit (threshold) via the negative binomial distribution, alarm is given if the actual value is bigger or equal than this threshold. `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 ago and  $\alpha=0.05$  fixed.
- "Bayes 2" reference values from 6 weeks ago and 13 weeks of the year ago (symmetrical around the comparable week). 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 Systems*


---

### Description

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

### 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 systems should be used with their parameters. The parameter funcName and range must be specified where funcName must be the appropriate method function (without 'algo.'). range defines the timepoints to be evaluated by the actual system. If control includes name this name is used in the survRes Object as name.

### Value

list of survRes Objects  
generated by the specified surveillance systems

### See Also

[algo.rki](#), [algo.bayes](#), [algo.farrington](#)

**Examples**

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

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

# this are some survResObjects
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 by the predictive version, i.e.

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

which corresponds to Equation 8-1 in the Farrington and Andrews chapter.

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

## Value

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

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

## Author(s)

M. Höhle

## Source

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

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

## See Also

[algo.rkiLatestTimepoint](#), [algo.bayesLatestTimepoint](#) and [algo.bayes](#) for the Bayes system.

## Examples

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

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

---

 algo.compare

---

*Comparison of Specified Surveillance Systems using Quality Values*


---

**Description**

Comparison of specified surveillance systems using quality values.

**Usage**

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

**Arguments**

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

**Value**

matrix           Matrix with values from [algo.quality](#), i.e. quality values for every surveillance system found in survResults.

**See Also**

[algo.quality](#)

**Examples**

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

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

algo.cusum

*CUSUM method*

### Description

Approximate one-side CUSUM method for a Poisson variate based on the cumulative sum of the deviation between a reference value  $k$  and the (standardized) observed values. An alarm is raised if the cumulative sum equals or exceeds a prespecified decision boundary  $h$ .

### Usage

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

### Arguments

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

**control** control object:

- range** determines the desired time points which should be evaluated
- k** is the reference value
- h** the decision boundary
- m** how to determine the expected number of cases – the following arguments are possible
  - numeric** a vector of values having the same length as `range`. If a single numeric value is specified then this value is replicated `length(range)` times.
  - NULL** A single value is estimated by taking the mean of all observations previous to the first `range` value.
  - "glm"** A GLM of the form

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

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

- trans** one of the following transformations (warning: `anscombe` and `negbin` transformations are experimental)
  - rossi** compute standardized variables  $z_3$  as proposed by Rossi
  - standard** compute standardized variables  $z_1$  (based on asymptotic normality)
  - anscombe** anscombe residuals – experimental
  - anscombe2nd** anscombe residuals as in Pierce and Schafer (1986) based on 2nd order approximation of  $E(X)$  – experimental

pearsonNegBin compute Pearson residuals for NegBin – experimental  
 anscombeNegBin anscombe residuals for NegBin – experimental  
 none no transformation  
 alpha parameter of the negative binomial distribution, s.t. the variance is  $m + \alpha * m^2$

### Details

This implementation is still experimental

### Value

survRes algo.cusum gives a list of class survRes which includes the vector of alarm values for every timepoint in range and the vector of cumulative sums for every timepoint in range for the system specified by k and h, the range and the input object of class disProg.  
 The upperbound entry shows for each time instance the number of diseased individuals it would have taken the cusum to signal. Once the CUSUM signals no resetting is applied, i.e. signals occurs until the CUSUM statistic again returns below the threshold.  
 The control\$*m.glm* entry contains the fitted glm object, if the original argument was "glm".

### Author(s)

M. Paul and M. Härdle

### References

G. Rossi, L. Lampugnani and M. Marchi (1999), An approximate CUSUM procedure for surveillance of health events, *Statistics in Medicine*, 18, 2111–2122  
 D. A. Pierce and D. W. Schafer (1986), Residuals in Generalized Linear Models, *Journal of the American Statistical Association*, 81, 977–986

### Examples

```

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

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

```

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

---

 algo.farrington

*Surveillance for a time series using the Farrington procedure.*


---

## Description

The function takes range values of the time series counts and for each uses a GLM to predict the number of counts according to the procedure by Farrington et. al. This is then compared to the observed number of counts and in case an exceedance of the confidence interval calculated is seen an alarm is raised.

## Usage

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

## Arguments

disProgObj	object of class disProgObj (including the observed and the state chain)
control	Control object <ul style="list-style-type: none"> <li>• range Specifies the index of all timepoints which should be tested. If range is NULL the maximum number of possible weeks is used.</li> <li>• b how many years back in time to include when forming the base counts.</li> <li>• w windows size, i.e. number of weeks to include before and after the current week</li> <li>• reweight Boolean specifying whether to perform reweight step</li> <li>• trend If true a trend is included and kept in case the conditions in the Farrington et. al. paper are met (see the results). If false then NO trend is fit.</li> <li>• verbose show extra debugging information</li> <li>• plot shows the final GLM model fit graphically (use History/Recording to see all pictures)</li> <li>• 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</li> <li>• alpha An approximate (two-sided) <math>(1 - \alpha)\%</math> prediction interval is calculated</li> <li>• 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 <math>cases = 5</math> reports were received in the past <math>period = 4</math></li> </ul>

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.

- `fitFunString` 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`. See [algo.farrington.fitGLM](#) for details.

## 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

An object of class `SurvRes`.

## Author(s)

M. HÃ¶hle

## Source

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

## See Also

[algo.farrington.fitGLM](#), [algo.farrington.threshold](#)

## Examples

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

#Do surveillance for the last 100 weeks.
n <- length(salmonella.agona$observed)
#Set control parameters.
control <- list(b=4,w=3,range=(n-100):n,reweight=TRUE, verbose=FALSE,alpha=0.01)
```

```
res <- algo.farrington(salmonella.agona,control=control)
#Plot the result.
plot(res,disease="Salmonella Agona",method="Farrington")

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

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

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

## End(Not run)
```

---

```
algo.farrington.assign.weights
      Assign weights to base counts
```

---

## Description

Weights are assigned according to the Anscombe residuals

## Usage

```
algo.farrington.assign.weights(s)
```

## Arguments

s                    Vector of standardized Anscombe residuals

## Value

Weights according to the residuals

## See Also

See Also as [anscombe.residuals](#)

---

algo.farrington.fitGLM

*Fit the Poisson GLM of the Farrington procedure for a single time point*

---

### Description

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

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

as specified by the Farrington procedure. That way we are able to predict the value  $c_0$ . 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)
```

### Arguments

response	The vector of observed base counts
wtime	Vector of week numbers corresponding to response
timeTrend	Boolean whether to fit the $\beta t$ or not
reweight	Fit twice – 2nd time with Anscombe residuals

### 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

*Threshold computations using a two sided confidence interval*

**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 (superflous?)
- alpha          Quantile level in Gaussian based CI, i.e. an  $(1 - \alpha)\%$  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 4 with lower and upper bounds of an  $(1 - \alpha)\%$  confidence interval (first two arguments) and corresponding quantile of observation y together with the median of the predictive distribution.

algo.glrnb

*Count data regression charts*

**Description**

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

**Usage**

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

## Arguments

<code>disProgObj</code>	object of class <code>disProg</code> to do surveillance for
<code>control</code>	A list controlling the behaviour of the algorithm
<code>range</code>	vector of indices in the observed vector to monitor (should be consecutive)
<code>mu0</code>	A vector of in-control values of the mean of the negative binomial distribution with the same length as <code>range</code> . If <code>NULL</code> the observed values in $1:(\min(\text{range})-1)$ are used to estimate beta through a generalized linear model. To fine-tune the model one can instead specify <code>mu0</code> as a list with two components: <ul style="list-style-type: none"> <li><code>S</code> number of harmonics to include</li> <li><code>trend</code> include a term <code>t</code> in the GLM model</li> </ul>
<code>alpha</code>	The (known) dispersion parameter of the negative binomial distribution. If <code>alpha=0</code> then the negative binomial distribution boils down to the Poisson distribution and a call of <code>algo.glrnb</code> is equivalent to a call to <code>algo.glrpois</code> . If <code>alpha=NULL</code> the parameter is calculated as part of the in-control estimation.
<code>c.ARL</code>	threshold in the GLR test, i.e. $c_\gamma$
<code>Mtilde</code>	number of observations needed before we have a full rank the typical setup for the "intercept" and "epi" charts is <code>Mtilde=1</code>
<code>M</code>	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 <code>M=-1</code> .
<code>change</code>	a string specifying the type of the alternative. Currently the two choices are <code>intercept</code> and <code>epi</code> . See the SFB Discussion Paper 500 for details.
<code>theta</code>	if <code>NULL</code> then the GLR scheme is used. If not <code>NULL</code> the prespecified value for $\kappa$ or $\lambda$ is used in a recursive LR scheme, which is faster.
<code>dir</code>	a string specifying the direction of testing in GLR scheme. With <code>"inc"</code> only increases in $x$ are considered in the GLR-statistic, with <code>"dec"</code> decreases are regarded.
<code>ret</code>	a string specifying the type of upperbound-statistic that is returned. With <code>"cases"</code> the number of cases that would have been necessary to produce an alarm or with <code>"value"</code> the glr-statistic is computed (see below).

## Details

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

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

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

So  $N$  is the time point where the GLR statistic is above the threshold the first time: An alarm is given and the surveillance is resetted starting from time  $N + 1$ . Note that the same c.ARL as before is used, but if  $\mu_0$  is different at  $N + 1, N + 2, \dots$  compared to time  $1, 2, \dots$  the run length properties differ. Because c.ARL to obtain a specific ARL can only be obtained by Monte Carlo simulation there is no good way to update c.ARL automatically at the moment. Also, FIR GLR-detectors might be worth considering.

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

## Value

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

## Author(s)

M. Hoehle

## Source

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

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

## See Also

[algo.rkiLatestTimepoint](#)

## Examples

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

```

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

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

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

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

```

---

algo.glrpois

*Poisson regression charts*

---

## Description

Poisson regression charts for the monitoring of surveillance time series.

## Usage

```

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

```

## Arguments

disProgObj	object of class disProg to do surveillance for
control	A list controlling the behaviour of the algorithm
range	vector of indices in the observed vector to monitor (should be consecutive)
mu0	A vector of in-control values of the Poisson distribution with the same length as range. If NULL the observed values in 1:(min(range)-1) are used to estimate beta through a generalized linear model. To fine-tune the model one can instead specify mu0 as a list with two components: <ul style="list-style-type: none"> <li>S number of harmonics to include</li> <li>trend include a term t in the GLM model</li> </ul>
c.ARL	threshold in the GLR test, i.e. $c_\gamma$

Mtilde number of observations needed before we have a full rank the typical setup for the "intercept" and "epi" charts is Mtilde=1  
M number of time instances back in time in the window-limited approach, i.e. the last value considered is  $\max(1, n - M)$ . To always look back until the first observation use M=-1.  
change a string specifying the type of the alternative. Currently the two choices are intercept and epi. See the SFB Discussion Paper 500 for details.  
theta if NULL then the GLR scheme is used. If not NULL the prespecified value for  $\kappa$  or  $\lambda$  is used in a recursive LR scheme, which is faster.  
dir a string specifying the direction of testing in GLR scheme. With "inc" only increases in  $x$  are considered in the GLR-statistic, with "dec" decreases are regarded.  
ret a string specifying the type of upperbound-statistic that is returned. With "cases" the number of cases that would have been necessary to produce an alarm or with "value" the glr-statistic is computed (see below).

**Details**

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

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

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

So  $N$  is the time point where the GLR statistic is above the threshold the first time: An alarm is given and the surveillance is resetted starting from time  $N + 1$ . Note that the same  $c$ .ARL as before is used, but if  $\mu_0$  is different at  $N + 1, N + 2, \dots$  compared to time  $1, 2, \dots$  the run length properties differ. Because  $c$ .ARL to obtain a specific ARL can only be obtained my Monte Carlo simulation there is no good way to update  $c$ .ARL automatically at the moment. Also, FIR GLR-detectors might be worth considering.

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

**Value**

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

**Author(s)**

M. Hoehle with contributions by V. Wimmer

**Source**

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

**See Also**

[algo.rkiLatestTimepoint](#)

**Examples**

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

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

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

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

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

**Description**

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

**Usage**

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

**Arguments**

disProgObj	object of class disProg
control	control object: <ul style="list-style-type: none"> <li>lambda If TRUE an autoregressive parameter <math>\lambda</math> is included, if lambda is a vector of logicals, unit-specific parameters <math>\lambda_i</math> are included. By default, observations <math>y_{t-lag}</math> at the previous time points, i.e. <math>lag = 1</math>, are used for the autoregression. Other lags can be used by specifying lambda as a vector of integers, see Examples and <a href="#">meanResponse</a> for details.</li> <li>neighbours If TRUE an autoregressive parameter for adjacent units <math>\phi</math> is included, if neighbours is a vector of logicals, unit-specific parameters <math>\phi_i</math> are included. By default, observations <math>y_{t-lag}</math> at the previous time points, i.e. <math>lag = 1</math>, are used for the autoregression. Other lags can be used by specifying neighbours as a vector of integers.</li> <li>linear a logical (or a vector of logicals) indicating whether a linear trend <math>\beta</math> (or a linear trend <math>\beta_i</math> for each unit) is included</li> <li>nseason Integer number of Fourier frequencies; if nseason is a vector of integers, each unit <math>i</math> gets its own seasonal parameters</li> <li>negbin if "single" negative binomial rather than poisson is used, if "multiple" unit-specific overdispersion parameters are used.</li> <li>proportion see details in <a href="#">meanResponse</a></li> <li>lag.range determines which observations are used to fit the model</li> </ul>
thetastart	vector with starting values for all parameters specified in the control object (for optim).
verbose	if true information about convergence is printed

**Details**

Note that for the time being this function is not a surveillance algorithm, but only a modelling approach as described in the Held et. al (2005) paper.

**Value**

ahg Returns an object of class ah with elements

- coefficientsestimated parameters
- seestimated standard errors
- covcovariance matrix
- loglikelihoodloglikelihood
- convergencellogical indicating whether optim converged or not
- fitted.valuesfitted mean values  $\mu_{i,t}$
- controlspecified control object
- disProgObjspecified disProg-object
- lagwhich lag was used for the autoregressive parameters  $\lambda$  and  $\phi$
- nObsnumber of observations used for fitting the model

### Author(s)

M. Paul, L. Held, M. Höhle

### Source

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

### See Also

[meanResponse](#)

### Examples

```
# univariate time series: salmonella agona cases
data(salmonella.agona)
salmonella <- create.disProg(week=1:length(salmonella.agona$observed),
                           observed=salmonella.agona$observed,
                           state=salmonella.agona$state)

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

algo.hhh(salmonella, control=model1)

# multivariate time series:
# measles cases in Lower Saxony, Germany
data(measles.weser)

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

# different starting values for
# theta = (lambda, beta, gamma_1, gamma_2, psi)
startValues <- c(0.1, rep(0, 3), 1)
algo.hhh(measles.weser, control=model1,
         thetastart=startValues)
```

```

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

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

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

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

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

algo.hhh(influMen, control=model2)

```

---

algo.hhh.grid

*Function to try multiple starting values*


---

## Description

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

## Usage

```

algo.hhh.grid(disProgObj, control=list(lambda=TRUE, neighbours=FALSE,
  linear=FALSE, nseason=0,
  negbin=c("none", "single", "multiple"),
  proportion=c("none", "single", "multiple"),lag.range=NULL),
  thetastartMatrix, maxTime=1800, verbose=FALSE)

```

**Arguments**

disProgObj	object of class disProg
control	control object: <ul style="list-style-type: none"> <li>• lambdaIf TRUE an autoregressive parameter <math>\lambda</math> is included, if lambda is a vector of logicals, unit-specific parameters <math>\lambda_i</math> are included. By default, observations <math>y_{t-lag}</math> at the previous time points, i.e. <math>lag = 1</math>, are used for the autoregression. Other lags can be used by specifying lambda as a vector of integers, see Examples and <a href="#">meanResponse</a> for details.</li> <li>• neighboursIf TRUE an autoregressive parameter for adjacent units <math>\phi</math> is included, if neighbours is a vector of logicals, unit-specific parameters <math>\phi_i</math> are included. By default, observations <math>y_{t-lag}</math> at the previous time points, i.e. <math>lag = 1</math>, are used for the autoregression. Other lags can be used by specifying neighbours as a vector of integers.</li> <li>• lineara logical (or a vector of logicals) indicating whether a linear trend <math>\beta</math> (or a linear trend <math>\beta_i</math> for each unit) is included</li> <li>• nseasoninteger number of Fourier frequencies; if nseason is a vector of integers, each unit <math>i</math> gets its own seasonal parameters</li> <li>• negbinif "single" negative binomial rather than poisson is used, if "multiple" unit-specific overdispersion parameters are used.</li> <li>• proportionsee details in <a href="#">meanResponse</a></li> <li>• lag.rangedetermines which observations are used to fit the model</li> </ul>
thetastartMatrix	matrix with initial values for all parameters specified in the control object as rows.
verbose	if true progress information is printed
maxTime	maximum of time (in seconds) to elapse until algorithm stops.

**Value**

ahg	Returns an object of class ahg with elements <ul style="list-style-type: none"> <li>• bestresult of a call to algo.hhh with highest likelihood</li> <li>• allLoglikvalues of loglikelihood for all starting values used</li> <li>• gridSize number of different starting values in thetastartMatrix</li> <li>• gridUsed number of used starting values</li> <li>• timeelapsed time</li> <li>• convergenceif false algo.hhh did not converge for all (used) starting values</li> </ul>
-----	---

**Author(s)**

M. Paul, L. Held

**Source**

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

**See Also**

[meanResponse](#), [create.grid](#), [algo.hhh](#)

**Examples**

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

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

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

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

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

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

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

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

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

# now meningococcal infections in the same week should enter as covariates
# (i.e. nu_flu,t = lambda_flu * y_flu,t-1
# and nu_men,t = lambda_men * y_men,t-1 + phi_men*y_flu,t )
```

```

model2 <- list(lambda=c(1,1), neighbours=c(NA,0),
              linear=FALSE,nseason=c(3,1),negbin="multiple")

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

## End(Not run)

```

---

algo.hmm

*Hidden Markov Model (HMM) method*


---

## Description

This function implements on-line HMM detection of outbreaks based on the retrospective procedure described in Le Strat and Carret (1999). Using the function `msm` (library `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))

```

## Arguments

<code>disProgObj</code>	object of class <code>disProg</code> (including the observed and the state chain)
<code>control</code>	control object:
	<code>range</code> 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 <code>range</code> should be high enough to allow for enough reference values for estimating the HMM
	<code>Mtilde</code> number of observations back in time to use for fitting the HMM (including current). Reasonable values are a multiple of <code>disProgObj\$freq</code> , the default is <code>Mtilde=-1</code> , which means to use all possible values - for long series this might take very long time!
	<code>noStates</code> number of hidden states in the HMM – the typical choice is 2. The initial rates are set such that the <code>noState</code> 'th state is the one having the highest rate. I.e. this state is considered the outbreak state.
	<code>trend</code> Boolean stating whether a linear time trend exists, i.e. if TRUE (default) then $\beta_j \neq 0$
	<code>noHarmonics</code> number of harmonic waves to include in the linear predictor. Default is 1.
	<code>covEffectEqual</code> see details
	<code>saveHMMs</code> Boolean, if TRUE then the result of the fitted HMMs is saved. With this option the function can also be used to analyse data retrospectively. Default option is FALSE

## Details

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

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

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

where

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

and  $nH = \text{noHarmonics}$  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  $j = 1, \dots, \text{noState}$ .

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

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

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

## Value

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

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

## Author(s)

M. HÃ¶hle

## References

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

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

**See Also**[msm](#)**Examples**

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

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

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

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

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

## End(Not run)
```

---

 algo.outbreakP

*Semiparametric surveillance of outbreaks*


---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

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

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

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

**Value**

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

**Author(s)**

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

**Source**

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

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

**References**

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

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

**Examples**

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

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

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

#Graphical illustration with number-needed-before-alarm (NNBA) upperbound.
res <- outbreakP(mysts, control=list(range=1:length(y),k=100,
                                     ret="cases"))
plot(res,dx.upperbound=0,lwd=c(1,1,3),legend.opts=list(legend=c("Infected","NNBA","Outbreak","Alarm"),horiz=TR
```

---

algo.quality

*Computation of Quality Values for a Surveillance System Result*

---

**Description**

Computation of the quality values for a surveillance System output.

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

list of quality values

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

**See Also**

[algo.compare](#)

**Examples**

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

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

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

---

 algo.rki

*The system used at the RKI*


---

## Description

Evaluation of timepoints with the detection algorithms used by the RKI

## Usage

```

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

## Arguments

disProgObj	object of class disProg (including the observed and the state chain).
timePoint	time point which 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**

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

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

**Author(s)**

M. Höhle, A. Riebler, Christian Lang

**See Also**

[algo.bayesLatestTimepoint](#) and [algo.bayes](#) for the Bayes system.

**Examples**

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

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

# The same for rki 1 to rki 3
algo.rki1(disProgObj, control = list(range = 200:208))
algo.rki2(disProgObj, control = list(range = 200:208))
algo.rki3(disProgObj, control = list(range = 200:208))

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

---

algo.rogerson

*Modified CUSUM method as proposed by Rogerson and Yamada (2004)*

---

**Description**

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

**Usage**

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

**Arguments**

**disProgObj** object of class `disProg` that includes a matrix with the observed number of counts

**control** list with elements

- range** vector of indices in the observed matrix of `disProgObj` to monitor
- theta0t** matrix with in-control parameter, must be specified
- ARL0** desired average run length  $\gamma$
- s** change to detect, see `findH` for further details
- hValues** matrix with decision intervals  $h$  for a sequence of values  $\theta_{0,t}$  (in the range of `theta0t`)
- distribution** "poisson" or "binomial"
- nt** optional matrix with varying sample sizes for the binomial CUSUM
- FIR** a FIR CUSUM with head start  $\frac{h}{2}$  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

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

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

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

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

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

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

**Value**

Returns an object of class survRes with elements

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

**Note**

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

**References**

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

**See Also**

[hValues](#)

**Examples**

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

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

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

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

---

algo.summary

*Summary Table Generation for Several Disease Chains*

---

**Description**

Summary table generation for several disease chains.

**Usage**

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

**Arguments**

compMatrices    list of matrices constructed by algo.compare.

**Details**

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

**Value**

matrix            summing up the singular input matrices

**See Also**

[algo.compare](#), [algo.quality](#)

**Examples**

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

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

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

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

---

 algo.twins

---

*Model fit based on a two-component epidemic model*


---

### Description

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

### Usage

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

### Arguments

disProgObj	object of class disProg
control	control object:
	burnin Number of burn in samples.
	filter Thinning parameter. If filter = 10 every 10th sample is after the burn in is returned.
	sampleSize Number of returned samples. Total number of samples = burnin+filter*sampleSize
	noOfHarmonics Number of harmonics to use in the modelling, i.e. $L$ in (2.2) of Held et al (2006).
	alpha_xi Parameter $\alpha_\xi$ of the hyperprior of the epidemic parameter $\lambda$
	beta_xi Parameter $\beta_\xi$ of the hyperprior of the epidemic parameter $\lambda$
	psiRWSigma Starting value for the tuning of the variance of the random walk proposal for the overdispersion parameter $\psi$ .
	alpha_psi Parameter $\alpha_\psi$ of the prior of the overdispersion parameter $\psi$
	beta_psi Parameter $\beta_\psi$ of the prior of the overdispersion parameter $\psi$
	nu_trend Adjust for a linear trend in the endemic part? (default: FALSE)
	logFile Base file name for the output files. The function writes three output files in your current working directory (i.e. getwd()). If logfile = "twins.log" the results are stored in the three files "twins.log", "twins.log2" and "twins.log.acc". "twins.log" contains the returned samples of the parameters $\psi$ , $\gamma_0$ , $\gamma_1$ , $\gamma_2$ , $K$ , $\xi_\lambda$ $\lambda_1, \dots, \lambda_n$ , the predictive distribution of the number of cases at time $n + 1$ and the deviance. "twins.log2" contains the sample means of the variables $X_t, Y_t, \omega_t$ and the relative frequency of a changepoint at time t for t=1,...,n and the relative frequency of a predicted changepoint at time n+1. "twins.log.acc" contains the acceptance rates of $\psi$ , the changepoints and the endemic parameters $\gamma_0, \gamma_1, \gamma_2$ in the third column and the variance of the random walk proposal for the update of the parameter $\psi$ in the second column.

**Details**

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

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

**Value**

Returns an object of class `atwins` with elements

<code>control</code>	specified control object
<code>disProgObj</code>	specified <code>disProg</code> -object
<code>logFile</code>	contains the returned samples of the parameters $\psi, \gamma_0, \gamma_1, \gamma_2, \mathbf{K}, \xi_\lambda, \lambda_1, \dots, \lambda_n$ , the predictive distribution and the deviance.
<code>logFile2</code>	contains the sample means of the variables $X_t, Y_t, \omega_t$ and the relative frequency of a changepoint at time $t$ for $t=1, \dots, n$ and the relative frequency of a predicted changepoint at time $n+1$ .

**Author(s)**

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

**References**

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

**Examples**

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

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

# Call algorithm and save result.
otwins <- algo.twins(hepatitisA)

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

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

---

anscombe.residuals	<i>Compute Anscombe residuals</i>
--------------------	-----------------------------------

---

**Description**

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

**Usage**

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

**Arguments**

<code>m</code>	<code>m</code> is a glm object of the fit
<code>phi</code>	<code>phi</code> is the current estimated over-dispersion

**Value**

Standardized Anscombe residuals of `m`

**References**

McCullagh & Nelder, Generalized Linear Models, 1989

---

ar1Cusum	<i>Calculation of Average Run Length for discrete CUSUM schemes</i>
----------	---

---

**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**

```
ar1Cusum(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 $\lambda$ 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 <code>digits</code> 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.

---

bestCombination	<i>Partition of a number into two factors</i>
-----------------	---

---

**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**

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

---

categoricalCUSUM	<i>CUSUM detector for time-varying categorical time series</i>
------------------	--

---

**Description**

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

**Usage**

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

**Arguments**

- |                      |  |
|----------------------|--|
| <code>stsObj</code>  | Object of class <code>sts</code> 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 <code>populationFrac</code> .  |
| <code>control</code> | Control object containing several items <ul style="list-style-type: none"> <li>• <code>rangeVector</code> of length <math>t_{max}</math> with indices of the observed slot to monitor.</li> <li>• <code>hThreshold</code> to use for the monitoring. Once the CUSUM statistics is larger or equal to <code>h</code> we have an alarm.</li> <li>• <math>\text{pi0}(k-1) \times t_{max}</math> in-control probability vector for all categories except the reference category.</li> <li>• <math>\text{mu1}(k-1) \times t_{max}</math> out-of-control probability vector for all categories except the reference category.</li> <li>• <code>dfun</code> The probability mass function or density used to compute the likelihood ratios of the CUSUM. In a negative binomial CUSUM this is <code>dnbinom</code>, in a binomial CUSUM <code>dbinom</code> and in a multinomial CUSUM <code>dmultinom</code>. The function must be able to handle the arguments <code>y</code>, <code>size</code>, <code>mu</code> and <code>log</code>. As a consequence, one in the case of the beta-binomial distribution has to write a small wrapper function.</li> <li>• <code>ret</code> Return the necessary proportion to sound an alarm in the slot upperbound or just the value of the CUSUM statistic. Thus, <code>ret</code> is one of the values in <code>c("cases", "value")</code>.</li> </ul> |
| <code>...</code>     | Additional arguments to send to <code>dfun</code> .  |

**Details**

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

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

**Value**

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

**Author(s)**

M. Höhle

**References**

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

**See Also**

[categoricalCUSUM](#)

**Examples**

```
#####
#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
require("gamlss")
phase1 <- 1:(2*52)
phase2 <- (max(phase1)+1) : nrow(abattoir)

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

#CUSUM parameters
```

```

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

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

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

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

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

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

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

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

#ToDo: Compute run length using LRCUSUM.runlength

```

---

CIdata	<i>Confidence-Interval for the Mean of the Poisson Distribution</i>
--------	---

---

**Description**

In the first column the mean from 0 to 20 is shown, In the second the lower and in the third the upper value of the 95 percent confidence interval. These intervals are used in the RKI Algorithms.

**Usage**

```
data(CIdata)
```

**Format**

A data frame with header.

**Source**

L. Sachs. Angewandte Statistik. Springer Verlag, 7. Auflage, S.446, 1991

**See Also**

[algo.rki](#)

**Examples**

```
require(surveillance)
data(CIdata)
```

---

compMatrix.writeTable	<i>Latex Table Generation</i>
-----------------------	-------------------------------

---

**Description**

generates a latex table

**Usage**

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

**Arguments**

compMatrix      Matrix which includes quality values for every surveillance system.

**Value**

xtable            Latex table of the entered matrix.

**Author(s)**

M. Höhle, A. Riebler, C. Lang

**Examples**

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

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

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

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

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

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

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

---

correct53to52

*Data Correction from 53 to 52 weeks*

---

**Description**

Correction of data from 53 to 52 weeks a year

**Usage**

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

**Arguments**

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

**Details**

[readData](#) reads data with 53 weeks a year, but normally one year is said to have 52 weeks.

**Value**

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

**See Also**

[readData](#)

**Examples**

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

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

---

create.disProg

*Creating an object of class disProg*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

disProg	object of class disProg
---------	-------------------------

**Author(s)**

M. Paul

**Examples**

```
# create an univariate disProg object for the salmonella.agona data
data(salmonella.agona)
week <- 1:length(salmonella.agona$observed)
salmonellaDisProg <- create.disProg(week=week, observed=salmonella.agona$observed,
state=salmonella.agona$state)

# plot salmonella cases
title <- "Salmonella Agona cases in the UK"
plot(salmonellaDisProg, title = title, xaxis.years=TRUE, legend.opts=NULL,
startyear = 1990, firstweek = 1)
```

---

create.grid

*Computes a matrix of initial values*

---

**Description**

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

**Usage**

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

**Arguments**

disProgObj	object of class disProg
control	specified model
params	list of parameters: param=c(lower, upper, length) <ul style="list-style-type: none"> <li>• epidemic autoregressive parameters <math>\lambda</math> and <math>\phi</math>.</li> <li>• endemic trend and seasonal parameters <math>\beta, \gamma_j</math>.</li> <li>• negbin overdispersion parameter for negative binomial model <math>\psi</math>.</li> </ul>

**Value**

matrix	matrix with gridSize starting values as rows
--------	--

**Author(s)**

M. Paul

**See Also**

[algo.hhh.grid](#)

**Examples**

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

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

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

---

deleval	<i>Surgical failures data</i>
---------	-------------------------------

---

## Description

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

## Usage

```
data(abattoir)
```

## Details

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

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

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

## References

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

De Leval, Marc R., Franois, 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.years=FALSE, ylab="Response", xlab="Patient number")
```

**Description**

Display methods for objects of class "sts".

**Details**

The plotting of time-series plots relies on two internal functions with `plot.sts.time` one being the work-horse. Its arguments are (almost) similar to `plot.survRes`. `k` is the column to plot.

```
plot.sts.time(x,
  type,method=x@control$name,disease=x@control$data,same.scale=TRUE,
  par.list=list(mfrow=magic.dim(nAreas),mar=par()$mar),...)
```

```
plot.sts.time.one <- function(x, k=1,
  domany=FALSE,ylim=NULL,xaxis.years=TRUE, xaxis.units=TRUE,
  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),alarm.symbol=list(pch=24,
  col=2, cex=1),cex=1,legend.opts=list(x="top",
  legend=NULL,lty=NULL,pch=NULL,col=NULL),dx.upperbound=0.5,
  hookFunc=function() {},...)
```

Note that the `hookFunc` is called after all the basic plotting has been done, i.e. it is not possible to control formatting with this function.

For spacetime plots the following internal function does all the work:

```
plot.sts.spacetime(x,type,legend=NULL,opts.col=NULL,
  labels=TRUE,wait.ms=250,cex.lab=0.7,verbose=FALSE,
  dev.printer=NULL,...)
```

**Alarmplot**

```
plot.sts.alarm <- function(x, lvl=rep(1,nrow(x)),
  ylim=NULL,xaxis.years=TRUE, xaxis.units=TRUE,
  epochsAsDate=x@epochAsDate, xlab="time", main=NULL,
  type="hhs",lty=c(1,1,2),col=c(1,1,4), outbreak.symbol = list(pch=3,
  col=3, cex=1),alarm.symbol=list(pch=24, col=2,
  cex=1),cex=1,cex.yaxis=1,...)
```

`print` is the method for printing `sts` objects.

**Value**

The methods are called for their side-effects.

**Usage**

```
plot(x,y,type,...) print(x,...)
```

**Arguments**

**x** an object of class "sts"

**y** missing

**type** a formula specifying the plot type, several options are possible:

`observed ~ time` The observations in `x` are aggregated over units and the resulting univariate time-series is plotted. The plotting is done by the function `plot.time.sts`, which takes the same arguments as the `plot.survRes` function.

`observed ~ time | unit` shows `dim(x)` plots with each showing the time-series of one observational unit. The actual plotting is done by the function `plot.time.sts.one`

`observed ~ 1 | unit` for each unit the counts are aggregated over time and a map illustrating the counts is shown. The column names of the `x@observed` object are used to label the entries of the `x@map`. Regions with an alarm are shaded.

`observed ~ 1 | unit * time` an animation consisting of `nrow(x)` frames is generated. Each frame contains the number of counts per region for the current row in the observed matrix. It is possible to redirect the output into files, e.g. to generate an animated GIF. See the examples.

`codealarm ~ time` Generates a so called alarmplot for a multivariate `sts` object. For each time point and each series it is shown whether there is an alarm. In case of hierarchical surveillance the user can pass an additional argument `lv1`, which is a vector of the same length as rows in `x` specifying for each time series its level.

... further arguments passed to or from other methods: in case of plotting these are passed to `plot`, in case of printing these are passed to `print.default`

**See Also**

[plot.survRes](#)

**Examples**

```
data(ha)
shp <- system.file("shapes/berlin.shp", package="surveillance")
has4 <- disProg2sts(ha, map=readShapePoly(shp, IDvar="SNAME"))

print(has4)
plot(has4, type= observed ~ time)
plot(has4, type= observed ~ time | unit)
plot(has4, type= observed ~ 1 | unit)
plot(has4[1:20,1:2], type= observed ~ 1 | unit)
plot(aggregate(has4,nfreq=13), type= observed ~ 1 | unit * time)

## Not run:
#Configure a png device printer for the plot command
dev.printer <-
```

```
list(device=png,extension=".png",width=640,height=480,name="/tmp/berlin")

#Do the animation
plot(aggregate(has4,nfreq=13), type = observed ~ 1 | unit * time,
      dev.printer=dev.printer)

#Use ImageMagick -- replace /sw/bin/convert by your path to convert
system(paste("/sw/bin/convert -delay 50 ",dev.printer$name,
             "*.png /tmp/animated.gif",sep=""))

## End(Not run)
```

---

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

<code>disProgObj</code>	object of class <code>disProg</code>
<code>map</code>	<code>SpatialPolygonsDataFrame</code> object containing the map visualization
<code>sts</code>	Object of class <code>sts</code> 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)))
```

---

`enlargeData`*Data Enlargement*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

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

**See Also**

[readData](#)

**Examples**

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

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

---

estimateGLRNbHook      *Hook function for in-control mean estimation*

---

### Description

Allows the user to specify his own estimation routine for the in-control mean of `algo.glrpois`  
Needs to work for `GLRNbHook`

### Usage

```
estimateGLRNbHook()
```

### Details

This hook function allows the user to customize the behaviour of the algorithm.

### Value

A list

<code>mod</code>	resulting model of a call of <code>glm.nb</code>
<code>range</code>	vector of length as <code>range</code> containing the predicted values

### Author(s)

M. Hoehle

### See Also

[algo.glrpois](#)

### Examples

```
## Not run:
estimateGLRNbHook <- function() {
  #Fetch control object from parent
  control <- parent.frame()$control
  #The period
  p <- parent.frame()$disProgObj$freq
  #Current range to perform surveillance on
  range <- parent.frame()$range

  #Define training & test data set (the rest)
  train <- 1:(range[1]-1)
  test <- range

  #Perform an estimation based on all observations before timePoint
  #Event better - don't do this at all in the algorithm - force
  #user to do it himself - coz its a model selection problem
```

```

data <- data.frame(y=parent.frame()$disProgObj$observed[t],t=train)
#Build the model equation
formula <- "y ~ 1 "
if (control$mu0Model$trend) { formula <- paste(formula," + t",sep="") }
for (s in 1:control$mu0Model$S) {
  formula <- paste(formula,"+cos(2*",s,"*pi/p*t)+ sin(2*",s,"*pi/p*t)",sep="")
}
#Fit the GLM
m <- eval(substitute(glm.nb(form,data=data),
                      list(form=as.formula(formula))))

#Predict mu_{0,t}
return(as.numeric(predict(m,newdata=data.frame(t=range),type="response")))
}

## End(Not run)

```

---

estimateGLRPoisHook     *Hook function for in-control mean estimation*

---

### Description

Allows the user to specify his own estimation routine for the in-control mean of algo.glrpois

### Usage

```
estimateGLRPoisHook()
```

### Details

This hook function allows the user to customize the behaviour of the algorithm.

### Value

A vector of length as range containing the predicted values.

### Author(s)

M. Hoehle

### See Also

[algo.glrpois](#)

**Examples**

```
## Not run:
estimateGLRPoisHook <- function() {
  #Fetch control object from parent
  control <- parent.frame()$control
  #The period
  p <- parent.frame()$disProgObj$freq
  #Current range to perform surveillance on
  range <- parent.frame()$range

  #Define training & test data set (the rest)
  train <- 1:(range[1]-1)
  test <- range

  #Perform an estimation based on all observations before timePoint
  #Event better - don't do this at all in the algorithm - force
  #user to do it himself - coz its a model selection problem
  data <- data.frame(y=parent.frame()$disProgObj$observed[t],t=train)
  #Build the model equation
  formula <- "y ~ 1 "
  if (control$mu0Model$trend) { formula <- paste(formula," + t",sep="") }
  for (s in 1:control$mu0Model$S) {
    formula <- paste(formula,"+cos(2*",s,"*pi/p*t)+ sin(2*",s,"*pi/p*t)",sep="")
  }
  #Fit the GLM
  m <- eval(substitute(glm(form,family=poisson(),data=data),list(form=as.formula(formula))))

  #Predict  $\mu_{0,t}$ 
  return(as.numeric(predict(m,newdata=data.frame(t=range),type="response")))
}

## End(Not run)
```

---

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)ccptance 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 <code>optim</code> . 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

```
find.kh(ARLa=500,ARLr=7,sided="one")
find.kh(ARLa=500,ARLr=3,sided="one")
```

---

findH	<i>Find decision interval for given in-control ARL and reference value</i>
-------	--

---

### Description

Function to find a decision interval  $h^*$  for given reference value  $k$  and desired ARL  $\gamma$  so that the average run length for a Poisson or Binomial CUSUM with in-control parameter  $\theta_0$ , reference value  $k$  and is approximately  $\gamma$ , 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 $\gamma$
theta0	in-control parameter $\theta_0$
s	change to detect, see details
distr	"poisson" or "binomial"

rel.tol	relative tolerance, i.e. the search for $h^*$ is stopped if $\left  \frac{ARL(h^*) - \gamma}{\gamma} \right  < \text{rel.tol}$
digits	the reference value $k$ and the decision interval $h$ are rounded to <code>digits</code> decimal places
roundK	passed to <code>findK</code>
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

<code>theta0</code>	in-control parameter
<code>h</code>	decision interval
<code>k</code>	reference value
<code>ARL</code>	ARL for a CUSUM with parameters $k$ and $h$
<code>rel.tol</code>	corresponds to $\left  \frac{ARL(h) - \gamma}{\gamma} \right $

---

<code>findK</code>	<i>Find reference value</i>
--------------------	-----------------------------

---

### Description

Calculates the reference value  $k$  for a Poisson or binomial CUSUM designed to detect a shift from  $\theta_0$  to  $\theta_1$

### 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.

---

ha	<i>Hepatitis A in Berlin</i>
----	------------------------------

---

**Description**

Number of Hepatitis A cases among adult male (age>18) in Berlin 2001-2006. An increase is seen during 2006

**Usage**

```
data(ha)
```

**Format**

A disProg object containing  $290 \times 12$  observations starting from week 1 in 2001 to week 30 in 2006.

**Source**

Robert Koch-Institut: SurvStat: <http://www3.rki.de/SurvStat>; Queried on 25 August 2006.

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

**Examples**

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

---

hepatitisA	<i>Hepatitis A in Germany</i>
------------	-------------------------------

---

**Description**

Weekly number of reported hepatitis A infections in Germany 2001-2004.

**Usage**

```
data(hepatitisA)
```

**Format**

A `disProg` object containing  $208 \times 1$  observations starting from week 1 in 2001 to week 52 in 2004.

**Source**

Robert Koch-Institut: SurvStat: <http://www3.rki.de/SurvStat>; Queried on 11 01 2005.

**Examples**

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

---

influMen	<i>Influenza and meningococcal infections in Germany, 2001-2006</i>
----------	---

---

**Description**

Weekly counts of new influenza and meningococcal infections in Germany 2001-2006.

**Usage**

```
data(influMen)
```

**Format**

A `disProg` object containing  $312 \times 2$  observations starting from week 1 in 2001 to week 52 in 2006.

**Source**

Robert Koch-Institut: SurvStat: <http://www3.rki.de/SurvStat>. Queried on 25 July 2007.

**Examples**

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

---

`isoWeekYear`*Find ISO week and ISO year of a vector of Date objects*

---

## Description

This function extracts the ISO week and ISO year of `Date` according to the ISO 8601 specification. Note that this function does nothing else than `format.Date(x, "` would do on Mac/Unix computers. However, this is not implemented on Windows.

A small internal wrapper for `format.Date` (called `formatDate`) thus directs all calls having one of these format strings to this function, if the `sessionInfo()[[1]]$os` information reveals a Windows system.

## Usage

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

## Arguments

Y	Date object (POSIX) or the year. Can be a vector.
M	month, NULL if Y is a Date object)
D	day, NULL if Y is a Date object)

## Details

The code is by Gustaf Rydevik <[gustaf.rydevik\\_at\\_gmail.com](mailto:gustaf.rydevik_at_gmail.com)> posted at <http://tolstoy.newcastle.edu.au/R/e10/help/10/05/55>

## Value

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

## Author(s)

Gustaf Rydevik

## Examples

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

---

loglikelihood	<i>Calculation of the loglikelihood needed in algo.hhh</i>
---------------	--

---

**Description**

Calculates the loglikelihood according to the model specified in designRes.

**Usage**

```
loglikelihood(theta, designRes)
```

**Arguments**

theta            vector of parameters

$$\theta = (\alpha_1, \dots, \alpha_m, \boldsymbol{\lambda}, \boldsymbol{\phi}, \boldsymbol{\beta}, \boldsymbol{\gamma}_1, \dots, \boldsymbol{\gamma}_m, \boldsymbol{\psi}),$$

where  $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_m)$ ,  $\boldsymbol{\phi} = (\phi_1, \dots, \phi_m)$ ,  $\boldsymbol{\beta} = (\beta_1, \dots, \beta_m)$ ,  $\boldsymbol{\gamma}_1 = (\gamma_{11}, \dots, \gamma_{1,2S_1})$ ,  $\boldsymbol{\gamma}_m = (\gamma_{m1}, \dots, \gamma_{m,2S_m})$ ,  $\boldsymbol{\psi} = (\psi_1, \dots, \psi_m)$ .

If the model specifies less parameters, those components are omitted.

designRes        Result of a call to make.design

**Value**

Returns the loglikelihood

**Author(s)**

M. Paul, L. Held

**See Also**

[meanResponse](#)

---

LRCUSUM.runlength	<i>Run length computation of a CUSUM detector</i>
-------------------	---

---

**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 <code>dnbinom</code> , in a binomial CUSUM <code>dbinom</code> and in a multinomial CUSUM <code>dmultinom</code> .
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 $M$ .
outcomeFun	A hook function to compute all possible outcome states to compute the likelihood ratio for. If NULL then the default function <code>outcomeFunStandard(k,n)</code> is used. This function uses the Cartesian product of $0:n$ for $k$ components.
...	Additional arguments to send to <code>dfun</code> .

**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.
ar1	If the model is time homogenous (i.e. if $T == 1$ ) then the ARL is computed based on the stationary distribution of the Markov chain. See the eqns in the reference for details.

**Author(s)**

M. Höhle

## References

- Höhle, M. (2010), Changepoint detection in categorical time series, Book chapter to appear in T. Kneib and G. Tutz (Eds.), Statistical Modelling and Regression Structures, Springer.
- Höhle, M. and Mazick, A. (2009), Aberration detection in R illustrated by Danish mortality monitoring, Book chapter to appear in T. Kass-Hout and X. Zhang (Eds.) Biosurveillance: A Health Protection Priority, CRCPress.
- Brook, D. and Evans, D. A. (1972), An approach to the probability distribution of Cusum run length, *Biometrika*, 59:3, pp. 539–549.

## See Also

[categoricalCUSUM](#)

## Examples

```
#####
#Run length of a time constant negative binomial CUSUM
#####

#In-control and out of control parameters
mu0 <- 10
alpha <- 1/2
kappa <- 2

#Density for comparison in the negative binomial distribution
dY <- function(y,mu,log=FALSE, alpha, ...) {
  dnbinom(y, mu=mu, size=1/alpha, log=log)
}

#In this case "n" is the maximum value to investigate the LLR for
#It is assumed that beyond n the LLR is too unlikely to be worth
#computing.
LRCUSUM.runlength( mu=t(mu0), mu0=t(mu0), mu1=kappa*t(mu0), h=5,
  dfun = dY, n=rep(100,length(mu0)), alpha=alpha)

h.grid <- seq(3,6,by=0.1)
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)$ar1
})
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 * 1:200) + 10,ncol=1)

r1 <- 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),r1$pmf,type="l",xlab="t",ylab="PMF")
plot(1:length(mu0),r1$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.
#####

```

---

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\_nrw '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\_nwrp 'Hepatitis A' in 'Oberbergischer Kreis, Olpe, Rhein-Sieg-kreis' (Germany, Nordrhein-Westfalen) and 'Siegenwittgenstein Altenkirchen' (Germany, Rheinland-Pfalz)

## Usage

```
data(m1)
```

## Format

disProg objects each containing 209 observations (weekly on 52 weeks)

**observed** Number of counts in the corresponding week

**state** Boolean whether there was an outbreak.

**Source**

Robert Koch-Institut: SurvStat: <http://www3.rki.de/SurvStat>; m1 and m3 were queried on 10 November 2004. The rest during September 2004.

**See Also**

[readData](#)

**Examples**

```
data(k1)
survResObj <- algo.rki1(k1, control=list(range=27:192))
plot(survResObj, "RKI 1", "k1", firstweek=27, startyear=2002)
```

---

magic.dim

*Returns a suitable k1 x k2 for plotting the disProgObj*

---

**Description**

For a given number k `magic.dim` provides a vector containing two elements `nRows` and `nCols` which can be used to set the dimension of a single graphic device so that `nRow*nCol` plots can be drawn by row (or by column) on the device.

**Usage**

```
magic.dim(k)
```

**Arguments**

k                    an integer

**Value**

vector with two elements

---

make.design	<i>Create the design matrices</i>
-------------	-----------------------------------

---

## Description

Creates the design matrices needed for meanResponse

## Usage

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

## Arguments

disProgObj	object of class disProg
control	control object:
	lambda If TRUE an autoregressive parameter $\lambda$ is included, if lambda is a vector of logicals, unit-specific parameters $\lambda_i$ are included. By default, observations $y_{t-lag}$ at the previous time points, i.e. $lag = 1$ , are used for the autoregression. Other lags can be used by specifying lambda as a vector of integers, see Examples and <a href="#">meanResponse</a> for details.
	neighbours If TRUE an autoregressive parameter for adjacent units $\phi$ is included, if neighbours is a vector of logicals, unit-specific parameters $\phi_i$ are included. By default, observations $y_{t-lag}$ at the previous time points, i.e. $lag = 1$ , are used for the autoregression. Other lags can be used by specifying neighbours as a vector of integers.
	linear a logical (or a vector of logicals) indicating whether a linear trend $\beta$ (or a linear trend $\beta_i$ for each unit) is included
	nseason Integer number of Fourier frequencies; if nseason is a vector of integers, each unit $i$ gets its own seasonal parameters
	negbin if "single" negative binomial rather than poisson is used, if "multiple" unit-specific overdispersion parameters are used.
	proportion see details in <a href="#">meanResponse</a>
	lag.range determines which observations are used to fit the model

## Value

list	<ul style="list-style-type: none"> <li>• Ymatrix with number of cases <math>y_{it}</math> in unit <math>i</math> at time <math>t</math> as elements, i.e. data without the first time point.</li> <li>• Ym1matrix with previous number of cases <math>y_{i,t-1}</math>, i.e data without the last time point.</li> <li>• Ym1.neighboursmatrix with weighted sum of earlier counts of adjacent units <math>\sum_{j \sim i} m_{ji} y_{j,t-1}</math></li> </ul>
------	--

- nOfNeighboursvector with number of neighbours for each unit  $i$
- X.trendSeasondesign matrix for linear trend and seasonal components
- populationFracmatrix with corresponding population proportions
- dimThetalist with number of parameters used in model
- controlcontrol object
- disProgObjObject of class disProg
- lagwhich lag is used for the autoregressive parameters  $\lambda$  and  $\phi$
- nObsnumber of observations

### Author(s)

M.Paul, L. Held

---

makePlot

*Plot Generation*

---

### Description

Just a test method.

### Usage

```
makePlot(outputpath, data = "k1", method = "rki1",
         name, disease, range = 157:339)
```

### Arguments

outputpath	path for the storage
data	abbreviation of the disease-file
method	method to be called
name	name of the method
disease	disease name
range	range to plot

### Details

makePlot reads the data given in data using the function readData, and the data are corrected to 52 weeks, enlarged using enlargeData and sent to the surveillance system given in method. The system result is plotted and stored in outputpath.

### Author(s)

M. Höhle, A. Riebler, C. Lang

**See Also**

[readData](#), [correct53to52](#), [enlargeData](#), [algo.call](#), [plot.survRes](#)

**Examples**

```
makePlot("./", "k1", "rki2", "RKI 2", "Kryptosporidiose")
```

---

meanResponse	<i>Calculate mean response needed in algo.hhh</i>
--------------	---

---

**Description**

Calculates the mean response for the model specified in designRes according to equations (1.2) and (1.1) in Held et al., 2005 for univariate time series and equations (3.3) and (3.2) (with extensions) for multivariate time series. See details.

**Usage**

```
meanResponse(theta, designRes)
```

**Arguments**

**theta** vector of parameters  
 $\theta = (\alpha_1, \dots, \alpha_m, \boldsymbol{\lambda}, \boldsymbol{\phi}, \boldsymbol{\beta}, \gamma_1, \dots, \gamma_m, \boldsymbol{\psi})$ ,  
 where  $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_m)$ ,  $\boldsymbol{\phi} = (\phi_1, \dots, \phi_m)$ ,  $\boldsymbol{\beta} = (\beta_1, \dots, \beta_m)$ ,  $\boldsymbol{\gamma}_1 = (\gamma_{11}, \dots, \gamma_{1,2S_1})$ ,  $\boldsymbol{\gamma}_m = (\gamma_{m1}, \dots, \gamma_{m,2S_m})$ ,  $\boldsymbol{\psi} = (\psi_1, \dots, \psi_m)$ .  
 If the model specifies less parameters, those components are omitted.

**designRes** Result of a call to `make.design`

**Details**

Calculates the mean response for a Poisson or a negative binomial model with mean

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

where

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

and  $\omega_j = 2\pi j / \text{period}$  are Fourier frequencies with known period, e.g. period=52 for weekly data, for a univariate time series.

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

The seasonal terms in the predictor can also be expressed as  $\gamma_s \sin(\omega_s t) + \delta_s \cos(\omega_s t) = A_s \sin(\omega_s t + \epsilon_s)$  with amplitude  $A_s = \sqrt{\gamma_s^2 + \delta_s^2}$  and phase difference  $\tan(\epsilon_s) = \delta_s / \gamma_s$ . The amplitude and

phase shift can be obtained from a fitted model by specifying `amplitudeShift=TRUE` in the `coef` method.

For multivariate time series the mean structure is

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

where

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

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

Alternatively, the mean can be specified as

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

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

## Value

Returns a list with elements

<code>mean</code>	matrix of dimension $n \times m$ with the calculated mean response for each time point and unit, where $n$ is the number of time points and $m$ is the number of units.
<code>epidemic</code>	matrix with the epidemic part $\lambda_i y_{i,t-1} + \phi_i \sum_{j \sim i} y_{j,t-1}$
<code>endemic</code>	matrix with the endemic part of the mean $n_{it} \nu_{it}$
<code>epi.own</code>	matrix with $\lambda_i y_{i,t-1}$
<code>epi.neighbours</code>	matrix with $\phi_i \sum_{j \sim i} y_{j,t-1}$

## Author(s)

M. Paul, L. Held

## Source

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

---

`measles.weser`*Measles epidemics in Lower Saxony in 2001-2002*

---

**Description**

Weekly counts of new measles cases for each "Kreis" of the administrative district "Weser-Ems" in Lower Saxony, Germany, in 2001 and 2002. All in all there are 15 "Kreise", two "Kreise" have been omitted

**Usage**

```
data(measles.weser)
```

**Format**

An multivariate object of class `disProg` with 104 observations for each one of the 15 Kreise.

**week** Number of week.

**observed** Matrix with number of counts in the corresponding week and Kreis.

**state** Boolean whether there was an outbreak – dummy not implemented.

**neighbourhood** Neighbourhood matrix.

**populationFrac** Population fractions.

**Examples**

```
data(measles.weser)
plot(measles.weser, as.one=FALSE)
```

---

`meningo.age`*Meningococcal infections in France 1985-1995*

---

**Description**

Monthly counts of meningococcal infections in France 1985-1995. Here, the data is split into 4 age groups (<1, 1-5, 5-20, >20).

**Usage**

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

**Format**

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

**week** Number of month

**observed** Matrix with number of counts in the corresponding month and age group

**state** Boolean whether there was an outbreak – dummy not implemented

**neighbourhood** Neighbourhood matrix, all age groups are adjacent

**populationFrac** Population fractions

**Source**

??

**Examples**

```
data(meningo.age)
plot(meningo.age, title="Meningococcal infections in France 1985-95")
plot(meningo.age, as.one=FALSE)
```

---

momo

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

---

**Description**

Weekly number of all cause mortality from 1994-2008 in each of the six age groups <1, 1-4, 5-14, 15-44, 45-64, 65-74, 75-84 and 85 years.

**Usage**

```
data(momo)
```

**Details**

The object of class `sts` contains the number of all cause mortality from 1994-2008 in Denmark for each of the six age groups <1, 1-4, 5-14, 15-44, 45-64, 65-74, 75-84 and 85 years. A special feature of such EuroMOMO data is that weeks are handled as defined by the ISO 8601 standard, which can be handled by the `sts` class.

The `population` slot of the `momo` object contains the population size in each of the six age groups. These are yearly data obtained from the StatBank Denmark.

The aim of the EuroMOMO project is to develop and strengthen real-time monitoring of mortality across Europe; this will enhance the management of serious public health risks such as pandemic influenza, heat waves and cold snaps. For further details see the homepage of the EuroMOMO project.

**Source**

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

**References**

Hjelle, M. and A. Mazick, A. (2009) Aberration detection in R illustrated by Danish mortality monitoring, Book chapter to appear in T. Kass-Hout and X. Zhang (Eds.) Biosurveillance: A Health Protection Priority, CRC Press.

EuroMOMO project page, <http://www.euromomo.eu/>, Last accessed: 13 Oct 2010.

**Examples**

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

---

observed-methods      *Methods*

---

**Description**

Method to extract or set the corresponding slot of an sts object. This documentation is not really valid yet.

**Methods**

`x = "sts"` The slot of x is determined and returned or set.

---

obsinyear-methods      *~~ Methods for Function obsinyear ~~*

---

**Description**

For each time point it gives the corresponding observation number within the year, e.g. the week or month number.

**Methods**

`x = "sts"` See above.

---

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. <ul style="list-style-type: none"> <li>• rangeVector of indices in the observed slot to monitor.</li> <li>• theta0 In-control parameters of the paired binary CUSUM.</li> <li>• theta1 Out-of-control parameters of the paired binary CUSUM.</li> <li>• h1 Primary control limit (=threshold) of 1st CUSUM.</li> <li>• h2 Primary control limit (=threshold) of 2nd CUSUM.</li> <li>• h11 Secondary limit for 1st CUSUM.</li> <li>• h22 Secondary limit for 2nd CUSUM.</li> </ul>
p	Vector giving the probability of the four different possible states, i.e. $c((\text{death}=0, \text{near-miss}=0), (\text{death}=1, \text{near-miss}=0), (\text{death}=0, \text{near-miss}=1), (\text{death}=1, \text{near-miss}=1))$ .
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 joining decision barriers
h22	together with h11 this makes up the joining decision barriers
sparse	Boolean indicating whether to use sparse matrix computations from the Matrix library (usually much faster!). Default: FALSE.

### Details

For details about the method see the Steiner et al. (1999) reference listed below. Basically, two individual CUSUMs are run based on a logistic regression model. The combined CUSUM not only signals if one of its two individual CUSUMs signals, but also if the two CUSUMs simultaneously cross the secondary limits.

**Value**

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

**Author(s)**

S. Steiner and M. Höhle

**References**

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

**See Also**

[categoricalCUSUM](#)

**Examples**

```
#Set in-control and out-of-control parameters as in paper
theta0 <- c(-2.3,-4.5,2.5)
theta1 <- c(-1.7,-2.9,2.5)

#Small helper function to compute the paired-binary likelihood
#of the length two vector yz when the true parameters are theta
dPBin <- function(yz,theta) {
  exp(dbinom(yz[1],size=1,prob=plogis(theta[1]),log=TRUE) +
    dbinom(yz[2],size=1,prob=plogis(theta[2]+theta[3]*yz[1]),log=TRUE))
}

#Likelihood ratio for all four possible configurations
p <- c(dPBin(c(0,0), theta=theta0), dPBin(c(0,1), theta=theta0),
      dPBin(c(1,0), theta=theta0), dPBin(c(1,1), theta=theta0))

#Compute ARL using non-sparse matrix operations
## Not run:
pairedbinCUSUM.runlength(p,w1=c(-1,37,-9,29),w2=c(-1,7),h1=70,h2=32,h11=38,h22=17)

## End(Not run)

#Sparse computations don't work on all machines (e.g. the next line
#might lead to an error. If it works this call can be considerably (!) faster
#than the non-sparse call.
## Not run:
pairedbinCUSUM.runlength(p,w1=c(-1,37,-9,29),w2=c(-1,7),h1=70,h2=32,
                        h11=38,h22=17,sparse=TRUE)

## End(Not run)

#Use paired binary CUSUM on the De Leval et al. (1994) arterial switch
#operation data on 104 newborn babies
data("deleval")
```

```

#Switch between death and near misses
observed(delevel) <- observed(delevel)[,c(2,1)]

#Run paired-binary CUSUM without generating alarms.
pb.surv <- pairedbinCUSUM(delevel,control=list(theta0=theta0,
      theta1=theta1,h1=Inf,h2=Inf,h11=Inf,h22=Inf))

plot(pb.surv, xaxis.years=FALSE)

#####
#Scale the plots so they become comparable to the plots in Steiner et
#al. (1999). To this end a small helper function is defined.
#####

#####
#Log LR for conditional specification of the paired model
#####
LLR.pairedbin <- function(yz,theta0, theta1) {
  #In control
  alphay0 <- theta0[1] ; alphaz0 <- theta0[2] ; beta0 <- theta0[3]
  #Out of control
  alphay1 <- theta1[1] ; alphaz1 <- theta1[2] ; beta1 <- theta1[3]
  #Likelihood ratios
  llry <- (alphay1-alphay0)*yz[1]+log(1+exp(alphay0))-log(1+exp(alphay1))
  llrz <- (alphaz1-alphaz0)*yz[2]+log(1+exp(alphaz0+beta0*yz[1]))-
      log(1+exp(alphaz1+beta1*yz[1]))
  return(c(llry=llry,llrz=llrz))
}

val <- expand.grid(0:1,0:1)
table <- t(apply(val,1, LLR.pairedbin, theta0=theta0, theta1=theta1))
w1 <- min(abs(table[,1]))
w2 <- min(abs(table[,2]))
S <- upperbound(pb.surv) / cbind(rep(w1,nrow(observed(pb.surv))),w2)

#Show results
par(mfcol=c(2,1))
plot(1:nrow(delevel),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(delevel),S[,2],type="l",main="Death",xlab="Patient No.",
      ylab="CUSUM Statistic")
lines(c(0,1e99), c(70,70),lty=2,col=2)
lines(c(0,1e99), c(38,38),lty=2,col=3)

#####
# Run the CUSUM with thresholds as in Steiner et al. (1999).
# After each alarm the CUSUM statistic is set to zero and

```

```

# monitoring continues from this point. Triangles indicate alarm
# in the respective CUSUM (nearmiss or death). If in both
# simultaneously then an alarm is caued by the secondary limits.
#####
pb.surv2 <- pairedbinCUSUM(delevel,control=list(theta0=theta0,
        theta1=theta1,h1=70*w1,h2=32*w2,h11=38*w1,h22=17*w2))

plot(pb.surv2, xaxis.years=FALSE)

```

---

plot.atwins

*Plot results of a twins model fit*


---

### Description

Plot results of fitting a twins model using MCMC output. Plots similar to those in the Held et al. (2006) paper are generated

### Usage

```

## S3 method for class 'atwins'
plot(x, which=c(1,4,6,7), ask=TRUE, ...)

```

### Arguments

x	An object of class atwins.
which	a vector containing the different plot types to show <ol style="list-style-type: none"> <li>1 A plot of the observed time series <math>Z</math> is shown together with posterior means for the number of endemic cases (<math>X</math>) and number of epidemic cases (<math>Y</math>).</li> <li>2 This plot shows trace plots of the gamma parameters over all MCMC samples.</li> <li>3 This shows a trace plot of <math>\psi</math>, which controls the overdispersion in the model.</li> <li>4 Autocorrelation functions for <math>K</math> and <math>\psi</math> are shown in order to judge whether the MCMC sampler has converged.</li> <li>5 Shows a plot of the posterior mean of the seasonal model <math>\nu[t]</math> together with 95% credibility intervals based on the quantiles of the posterior.</li> <li>6 Histograms illustrating the posterior density for <math>K</math> and <math>\psi</math>. The first one corresponds to Fig. 4(f) in the paper.</li> <li>7 Histograms illustrating the predictive posterior density for the next observed number of cases <math>Z[n+1]</math>. Compare with Fig.5 in the paper.</li> </ol>
ask	Boolean indicating whether to ask for a newline before showing the next plot.
...	Additional control for the plots, which are currently ignored.

### Details

For details see the plots in the paper. Basically MCMC output is visualized. This function is together with `algo.twins` still experimental.

**Value**

This function does not return anything.

**Author(s)**

M. Hofmann and M. Höhle

**References**

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

**See Also**

[algo.twins](#)

**Examples**

```
## Not run:
##Apparently, the algo.atwins can crash on some LINUX systems
##thus for now the example section is commented

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

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

#Call algorithm and save result
otwins <- algo.twins(hepatitisA)

#This shows the entire output
plot(otwins,which=c(1,2),ask=FALSE)

## End(Not run)
```

---

plot.disProg

*Plot Generation of the Observed and the defined Outbreak States of a  
(multivariate) time series*

---

**Description**

Plotting of a disProg object.

**Usage**

```
## S3 method for class 'disProg'
plot(x, title = "", xaxis.years=TRUE, startyear = x$start[1],
     firstweek = x$start[2], as.one=TRUE, same.scale=TRUE, ...)
## S3 method for class 'disProg.one'
plot(x, title = "", xaxis.years=TRUE, quarters=TRUE,
     startyear = x$start[1], firstweek = x$start[2], ylim=NULL, xlab="time",
     ylab="No. infected", type="hh", lty=c(1,1), col=c(1,1),
     outbreak.symbol = list(pch=3, col=3), legend.opts=list(x="top",
     legend=c("Infected", "Outbreak"), lty=NULL, pch=NULL, col=NULL), ...)
```

**Arguments**

x	object of class disProg
title	plot title
xaxis.years	if TRUE, the x axis is labeled using years
quarters	add quarters to the plot
startyear	year to begin the axis labeling (the year where the oldest data come from). This arguments will be obsolete in sts.
firstweek	number of the first week of January in the first year (just for axis labeling grounds)
as.one	if TRUE all individual time series are shown in one plot
same.scale	if TRUE all plots have same scale
ylim	range of y axis
xlab	label of the x-axis
ylab	label of the y-axis
type	line type of the observed counts (should be hh)
lty	line type of the observed counts
col	color of the observed count lines
outbreak.symbol	list with entries pch and col specifying the plot symbol
legend.opts	a list containing the entries to be sent to the <a href="#">legend</a> function. If no legend is requested use legend.opts=NULL. Otherwise, the following arguments are default <ul style="list-style-type: none"> <li>x top</li> <li>legend The names infected and outbreak</li> <li>lty If NULL the lty argument will be used</li> <li>pch If NULL the pch argument is used</li> <li>col If NULL the col argument is used</li> </ul> <p>An further arguments to the legend function are just provided as additional elements of this list, e.g. horiz=TRUE.</p>
...	further arguments for the function matplot

**Value**

a plot showing the number of infected and the defined alarm status for a time series created by simulation or given in data either in one single plot or in several plots for each individual time series.

**Author(s)**

M. Höhle with contributions by A. Riebler and C. Lang

**Examples**

```
# Plotting of simulated data
disProgObj <- sim.pointSource(p = 0.99, r = 0.5, length = 208,
                             A = 1, alpha = 1, beta = 0, phi = 0,
                             frequency = 1, state = NULL, K = 5)
# plot the simulated disease with the defined outbreaks
plot(disProgObj)
title <- "Number of Infected and Defined Outbreak Positions for Simulated Data"
plot(disProgObj, title = title)
plot(disProgObj, title = title, xaxis.years=TRUE,
      startyear = 1999, firstweek = 13)
plot(disProgObj, title = title, xaxis.years=TRUE,
      startyear = 1999, firstweek = 14)

# Plotting of measles data
data(measles.weser)
# one plot
plot(measles.weser, title = "measles cases in the district Weser-Ems",
      xaxis.years=TRUE, startyear= 2001, firstweek=1)
# plot cases for each "Kreis"
plot(measles.weser, same.scale=TRUE, as.one=FALSE)
```

---

plot.survRes

*Plot a survRes object*

---

**Description**

Plotting of a (multivariate) survRes object. The function plot.survRes.one is used as a helper function to plot a univariate time series.

**Usage**

```
## S3 method for class 'survRes'
plot(x, method=x$control$name, disease=x$control$data,
      xaxis.years=TRUE, startyear = 2001, firstweek = 1, same.scale=TRUE,...)
## S3 method for class 'survRes.one'
plot(x, method=x$control$name, disease=x$control$data,
      domany=FALSE, ylim=NULL, xaxis.years=TRUE, startyear = 2001, firstweek = 1,
```

```
xlab="time", ylab="No. infected", main=NULL, type="hhs",
lty=c(1,1,2),col=c(1,1,4),
outbreak.symbol = list(pch=3,col=3),alarm.symbol=list(pch=24,col=2),
legend.opts=list(x="top",
legend=c("Infected", "Upperbound", "Alarm", "Outbreak"),
lty=NULL,col=NULL,pch=NULL), ...)
```

### Arguments

x	object of class <code>survRes</code>
method	surveillance method to be used in title
disease	name of disease in title
xaxis.years	Boolean indicating whether to show a year based x-axis for weekly data
domany	Boolean telling the function whether it is called for a multivariate (TRUE) or univariate (FALSE) <code>survRes</code> object. In case of TRUE no titles are drawn.
ylim	range of y axis
startyear	year to begin the axis labeling (the year where the oldest data come from)
firstweek	number of the first week of January in the first year (just for axis labeling reasons)
xlab	label of the x-axis
ylab	label of the y-axis
main	the title of the graphics is generated from the <code>method</code> and <code>disease</code> arguments if not specified otherwise
same.scale	plot all time series with the same <code>ylim</code> ? Defaults to <code>true</code> .
type	line type of the observed counts (first two elements) and the upper bound (third element)
lty	vector of size 3 specifying the line type of the observed counts (left, right) and the upperbound line
col	vector with three elements: color of left bar and color of top bar, color of right bar, col of the upperbound line.
outbreak.symbol	list with entries <code>pch</code> and <code>col</code> specifying the plot symbol
alarm.symbol	list with entries <code>pch</code> and <code>col</code> specifying the plot symbol
legend.opts	a list containing the entries to be sent to the <a href="#">legend</a> function. If no legend is requested use <code>legend.opts=NULL</code> . Otherwise, the following arguments are default <code>x top</code> <code>legend</code> The names <code>infected</code> and <code>outbreak</code> . <code>lty</code> If <code>NULL</code> the <code>lty</code> argument will be used <code>pch</code> If <code>NULL</code> the <code>pch</code> argument is used <code>col</code> If <code>NULL</code> the <code>col</code> argument is used Any further arguments to the <code>legend</code> function are just provided as additional elements of this list, e.g. <code>horiz=TRUE</code> .
...	further arguments for the function <code>matplot</code> . If e.g. <code>xlab</code> or <code>main</code> are provided they overwrite the default values.

**Details**

The `plot.survRes.one` is intended for internal use. At the moment none of the surveillance methods support multivariate `survRes` objects. New versions of the packages currently under development will handle this.

**Value**

none. A plot showing the number of infected, the threshold for recognizing an outbreak, the alarm status and the outbreak status is generated.

**Author(s)**

M. Höhle

**Examples**

```
data(ha)
ctrl <- list(range = 209:290, b = 2, w = 6, alpha = 0.005)
plot(algo.bayes(aggregate(ha), control = ctrl))
```

---

predict.ah

*Predictions from a HHH model*

---

**Description**

Use a `ah` or `ahg` object for prediction.

**Usage**

```
## S3 method for class 'ah'
predict(object,newdata=NULL,
        type=c("response","endemic","epi.own","epi.neighbours"), ...)
```

**Arguments**

<code>object</code>	object of class <code>ah</code> or <code>ahg</code>
<code>newdata</code>	optionally, a <code>disProgObject</code> with which to predict; if omitted, the fitted mean is returned.
<code>type</code>	the type of prediction required. The default is on the scale of the response variable (endemic and epidemic part). The alternative "endemic" returns only the endemic part (i.e. $n_{it}\nu_{it}$ ), "epi.own" and "epi.neighbours" return the epidemic part (i.e. $\lambda_i y_{i,t}$ and $\phi_i \sum_{j \sim i} y_{j,t-1}$ )
<code>...</code>	not really used

**Details**

this function is still experimental

**Value**

matrix of values containing the mean  $\mu_{it}$  for each region and time point.

---

primeFactors	<i>Prime number factorization</i>
--------------	-----------------------------------

---

**Description**

Computes prime number factorization of an integer x.

**Usage**

```
primeFactors(x)
```

**Arguments**

x	an integer
---	------------

**Value**

vector with prime number factorization of x

---

print.algoQV	<i>Print quality value object</i>
--------------	-----------------------------------

---

**Description**

Print a single quality value object in a nicely formatted way

**Usage**

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

**Arguments**

x	Quality Values object generated with quality
...	Further arguments (not reall used)

**Examples**

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

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

# Compute the quality values in a nice formatted way
algo.quality(survResObj)
```

---

readData

*Reading of Disease Data*


---

**Description**

Reading of disease data. In the package disease data are saved in a file <abb>.txt containing three columns – the weeknumber (week), the observed number of counts (observed) and a state (state). The data are read using `read.table(..., header=T)`, hence the file has to contain a header.

**Usage**

```
readData(abb, week53to52=TRUE, sysPath=TRUE)
```

**Arguments**

abb	abbreviation of the diseasename.
week53to52	Boolean indicating whether to convert RKI 53 Weeks System to 52 weeks a year
sysPath	Boolean, if TRUE then R automatically looks in the data directory of the surveillance package.

**Details**

This function is only kept for backwards compability. As of 0.9-2 all data should be read with `data`.

**Value**

disProg	a object disProg (disease progress) including a list of the observed and the state chain.
---------	---

**See Also**

[m1](#), [m2](#), [m3](#), [m4](#), [m5](#), [q1\\_nrwh](#), [q2](#), [s1](#), [s2](#), [s3](#), [k1](#), [n1](#), [n2](#), [h1\\_nrwrp](#)

**Examples**

```
readData("m5")

#To bring a single vector of counts into a format, which can be
#handled by readData. Assume "counts" is a vector of counts.
counts <- rpois(100,20)
counts <- data.frame("week"=1:length(counts),"observed"=counts,
                    "state"=rep(0,length(counts)))
write(c("week","observed","state"),file="disease.txt",ncol=3)
write(t(as.matrix(counts)),file="disease.txt",ncol=3,append=TRUE)
disease <- readData("disease",week53to52=FALSE,sysPath=FALSE)
```

---

refvalIdxByDate	<i>Compute indices of reference value using Date class</i>
-----------------	--

---

**Description**

The reference values are formed base 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

---

residuals.ah	<i>Residuals from a HHH model</i>
--------------	-----------------------------------

---

**Description**

Extracts model residuals from a ah or ahg object.

**Usage**

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

**Arguments**

object	object of class ah or ahg
type	the type of residuals which should be returned. The alternatives are "deviance" (default) and "pearson"
...	not really used

**Details**

this function is still experimental

**Value**

matrix with residuals for each region and time point.

---

salmonella.agona	<i>Salmonella Agona cases in the UK 1990-1995</i>
------------------	---

---

**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 data frame with 312 observations on the following 2 variables.

**week** First four digits are the year, last two the week number within that year

**observed** Number of counts in the corresponding week

**state** Boolean whether there was an outbreak – dummy not implemented.

**Source**

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

**Examples**

```
data(salmonella.agona)
plot(salmonella.agona$observed,type="l",ylab="counts",xlab="")
```

---

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  $295 \times 1$  observations starting from week 1 in 2001 to week 35 in 2006.

**Source**

Robert Koch-Institut: SurvStat: <http://www3.rki.de/SurvStat>; Queried on September 2006.

Robert Koch Institut, Epidemiologisches Bulletin 31/2006.

**Examples**

```
data(shadar)
plot(shadar)
```

---

sim.pointSource      *Generation of Simulated Point Source Epidemy*

---

### Description

Simulation of epidemics which were introduced by point sources. The basis of this programme is a combination of a Hidden Markov Modell (to get random timepoints for outbreaks) and a simple model (compare [sim.seasonalNoise](#)) to simulate the epidemy.

### Usage

```
sim.pointSource(p = 0.99, r = 0.01, length = 400, A = 1,
               alpha = 1, beta = 0, phi = 0, frequency = 1, state = NULL, K)
```

### Arguments

p	probability to get a new epidemy at time i if there was one at time i-1, default 0.99.
r	probability to get no new epidemy at time i if there was none at time i-1, default 0.01.
length	number of weeks to model, default 400. length is ignored if state is given. In this case the length of state is used.
A	amplitude (range of sinus), default = 1.
alpha	parameter to move along the y-axis (negative values not allowed) with $\alpha > = A$ , default = 1.
beta	regression coefficient, default = 0.
phi	factor to create seasonal moves (moves the curve along the x-axis), default = 0.
frequency	factor to determine the oscillation-frequency, default = 1.
state	use a state chain to define the status at this timepoint (outbreak or not). If not given a Markov chain is generated by the programme, default NULL.
K	additional weight for an outbreak which influences the distribution parameter mu, default = 0.

### Value

disProg      a object disProg (disease progress) including a list of the observed, the state chain and nearly all input parameters.

### Author(s)

M. Höhle, A. Riebler, C. Lang

### See Also

[sim.seasonalNoise](#)

**Examples**

```
# Plotting of simulated data
disProgObj <- sim.pointSource(p = 0.99, r = 0.5, length = 208,
                             A = 1, alpha = 1, beta = 0, phi = 0,
                             frequency = 1, state = NULL, K = 2)
# plot the simulated disease with the defined outbreaks
plot(disProgObj)

state <- rep(c(0,0,0,0,0,0,0,0,0,1,1), 20)
disProgObj <- sim.pointSource(state = state, K = 1.2)
plot(disProgObj)
```

---

sim.seasonalNoise      *Generation of Background Noise for Simulated Timeseries*

---

**Description**

Generation of a cyclic model of a Poisson distribution as background data for a simulated timevector.

The mean of the Poisson distribution is modelled as:

$$\mu = \exp(A \sin(\text{frequency} \cdot \omega \cdot (t + \phi)) + \alpha + \beta * t + K * \text{state})$$

**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**

seasonNoise	Object of class seasonNoise which includes the modelled timevector, the parameter mu and all input parameters.
-------------	--

**Author(s)**

M. Höhle, A. Riebler, C. Lang

**See Also**

[sim.pointSource](#)

**Examples**

```
season <- sim.seasonalNoise(length = 300)
plot(season$seasonalBackground, type = "l")

# use a negative timetrend beta
season <- sim.seasonalNoise(beta = -0.003, length = 300)
plot(season$seasonalBackground, type = "l")
```

---

simHHH

*Simulates data based on the model proposed by Held et. al (2005)*

---

**Description**

Simulates a multivariate time series of counts based on the Poisson/Negative Binomial model as described in Held et al. (2005).

**Usage**

```
## Default S3 method:
simHHH(model=NULL, control = list(coefs = list(alpha=1, gamma = 0, delta = 0,
  lambda = 0, phi = NULL, psi = NULL, period = 52),
  neighbourhood = NULL, population = NULL, start = NULL),

  length)

## S3 method for class 'ah'
simHHH(model, control = model$control, length)
```

**Arguments**

**control** list with

- coefs** list with the following parameters of the model - if not specified, those parameters are omitted
- alpha** vector of length  $m$  with intercepts for  $m$  units or geographic areas respectively
- gamma** vector with parameters for the "sine" part of  $\nu_{i,t}$
- delta** vector with parameters for the "cosine" part of  $\nu_{i,t}$

	<b>lambda</b> autoregressive parameter
	<b>phi</b> autoregressive parameter for adjacent units
	<b>psi</b> overdispersion parameter of the negative binomial model; NULL corresponds to a Poisson model
	<b>period</b> period of the seasonal component, defaults to 52 for weekly data
	<b>neighbourhood</b> neighbourhood matrix of size $m \times m$ with element 1 if two units are adjacent; the default NULL assumes that there are no neighbours
	<b>population</b> matrix with population proportions; the default NULL sets $n_{i,t} = 1$
	<b>start</b> if NULL, the means of the endemic part in the $m$ units is used as initial values $y_{i,0}$
model	Result of a model fit with <a href="#">algo.hhh</a> , the estimated parameters are used to simulate data
length	number of time points to simulate

### Details

Simulates data from a Poisson or a Negative Binomial model with mean

$$\mu_{it} = \lambda y_{i,t-1} + \phi \sum_{j \sim i} y_{j,t-1} + n_{it} \nu_{it}$$

where

$$\log \nu_{it} = \alpha_i + \sum_{s=1}^S (\gamma_s \sin(\omega_s t) + \delta_s \cos(\omega_s t))$$

$\omega_s = 2s\pi/\text{period}$  are Fourier frequencies and  $n_{it}$  are possibly standardized population sizes.

### Value

Returns a list with elements

data	disProgObj of simulated data
mean	matrix with mean $\mu_{i,t}$ that was used to simulate the data
endemic	matrix with only the endemic part $\nu_{i,t}$
coefs	list with parameters of the model

### Note

The model does not contain a linear trend.

### Source

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

---

stcd *Spatio-temporal cluster detection*

---

### Description

Shiryaev-Roberts based prospective spatio-temporal cluster detection as in Assuncao & Correa (2009).

### Usage

```
stcd(x, y,t,radius,epsilon,areaA, areaAcapBk, threshold, cusum=FALSE)
```

### Arguments

x	Vector containing spatial x coordinate of the events.
y	Vector containing spatial y coordinate of the events.
t	Vector containing the time points of the events. It is assumed that the vector is sorted (early->last).
radius	Radius of the cluster to detect.
epsilon	Relative change of event-intensity within the cluster to detect. See reference paper for an explicit definition.
areaA	???
areaAcapBk	???
threshold	Threshold limit for the alarm and should be equal to the desired Average-Run-Length (ARL) of the detector.
cusum	???

### 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, UMFG, Brazil and provided by Thais Correa, UMFG, 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**

```
library("splancs")
data(burkitt)

# order the times
burkitt <- burkitt[order(burkitt$t), ]

#Parameters for the SR detection
epsilon <- 0.5 # relative change within the cluster
radius <- 20 # radius
threshold <- 161 # threshold limit

res <- stcd(x=burkitt$x,
            y=burkitt$y,
            t=burkitt$t,
            radius=radius,
            epsilon=epsilon,
            areaA=1,
            areaAcapBk=1,
            threshold=threshold)

#Index of the event
which.max(res$R >= threshold)
```

sts-class

Class "sts" – surveillance time series

**Description**

This is a rather lightweight class to implement multivariate time series of count used for public health surveillance data. The class captures the time series data as well as the spatial layout of the regions, where the data originate from.

**Slots**

- week:** Object of class `numeric` specifying the week numbers. Actually this is not really used at the moment.
- freq:** If weekly data `freq` corresponds to 52, in case of monthly data `freq` is 12.
- start:** vector of length two denoting the year and the sample number (week, month, etc.) of the first observation
- observed:** A matrix of size `length(week)` times the number of regions containing the weekly/monthly number of counts in each region. The colnames of the matrix should match the ID values of the shapes in the `map` slot.
- state:** Matrix with the same dimension as `observed` containing booleans whether at the specific time point there was an outbreak in the region
- alarm:** Matrix with the same dimension as `observed` specifying whether an outbreak detection algorithm declared a specific time point in the region as having an alarm. If the object contains just observations then this slot is null.
- upperbound:** Matrix with upper bound values
- neighbourhood:** Symmetric matrix of booleans size  $(\text{number of regions})^2$  stating the neighbourhood matrix.
- populationFrac:** Object of class `matrix`.
- map:** Object of class `SpatialPolygonsDataFrame` providing a shape of the areas which are monitored.
- control:** Object of class `list`, this is a rather free data type to be returned by the surveillance algorithms.
- epochAsDate:** Object of class `"logical"` stating whether to use a ISO 8601 representation of the epoch/week slot using the `Date` class (`epochAsDate=TRUE`) or just to interpret the epochs/weeks as numerics (`epochAsDate=FALSE`).
- multinomialTS:** Object of class `"logical"` stating whether to interpret the object as observed out of population, i.e. a multinomial interpretation instead of a count interpretation.

**Methods**

- nrow** signature(`x = "sts"`): extract number of rows of the observed matrix slot. The dimension of the other matrix slots is similar.
- ncol** signature(`x = "sts"`): extract number of columns of the observed matrix slot.

- dim** signature(x = "sts"): extract matrix dimensions of observed using [dim](#).
- observed** signature(x = "sts"): extract the observed slot of an sts object.
- population** signature(x = "sts"): extract the population slot of an sts object.
- alarms** signature(x = "sts"): extract the alarm slot of an sts object.
- upperbound** signature(x = "sts"): extract the upperbound slot of an sts object.
- control** signature(x = "sts"): extract the control slot of an sts object.
- epoch** signature(x = "sts"): extract the epoch slot of an sts object. If ISO dates are used then the returned object is of class Date.
- epochInYear** signature(x = "sts"): Returns the epoch number within the year of the epoch slot.
- colnames** signature(x="sts",do.NULL="missing",prefix="missing"): extract [colnames](#) of the observed matrix.
- initialize** signature(x="sts"): the internal function `init.sts` is called, which assigns all slots.
- aggregate** signature(x="sts"): see [aggregate, sts-method](#)
- year** signature(x = "sts"): extracts the corresponding year of each observation of x
- obsinyear** signature(x = "sts"): extracts the corresponding week number within the year of each observation of x
- as.data.frame** signature(x = "sts"): converts the observed, week, state and alarm slots of x into a data frame with column names matching the colnames of the respective slots. Useful when one wants to fit a model based on the object
- plot** signature(x="sts",y="missing",function(x, y, type,...)...): this function is the successor of the [plot.disProg](#) and [plot.survRes](#) functions. It takes (more or less) the same arguments as `plot.survRes`. The most important difference is the type of plot, which is specified using `type`. See [show, sts-method](#) for details.

**Author(s)**

M. Höhle

**Examples**

```
data("ha")
shp <- system.file("shapes/berlin.shp",package="surveillance")
ha <- disProg2sts(ha, map=readShapePoly(shp, IDvar="SNAME"))
plot(ha,type=observed ~ 1 | unit)
```

---

sumNeighbours	<i>Calculates the sum of counts of adjacent areas</i>
---------------	---

---

**Description**

Calculates the sum of counts of adjacent units/areas, i.e.  $\sum_{j \sim i} y_{j,t}$  for all time points  $t$  and each unit  $i, t = 1, \dots, n, i = 1, \dots, m$ .

**Usage**

```
sumNeighbours(disProgObj)
```

**Arguments**

disProgObj	Object of class disProg
------------	-------------------------

**Value**

matrix of dimension  $n \times m$

---

test	<i>Print xtable for several diseases and the summary</i>
------	--

---

**Description**

Just a test method

**Usage**

```
test(data = c("k1", "m5"), range = 157:339)
```

**Arguments**

data	vector of abbreviations for the diseases
range	timepoints to evaluate

**Details**

The specified datasets are readed, corrected, enlarged and sent to the RKI 1, RKI 2, RKI 3 and Bayes system. The quality values are computed and printed for each disease as latex table. Additionally a summary latex table for all diseases is printed

**Value**

xtable	printed latex tables
--------	----------------------

**Author(s)**

M. Höhle, A. Riebler, C. Lang

**Examples**

```
test(c("m1", "m2", "m3", "m4", "m5", "q1_nrwh", "q2", "s1",
      "s2", "s3", "k1", "n1", "n2", "h1_nrwrp"))
```

---

testSim

*Print xtable for a Simulated Disease and the Summary*

---

**Description**

Just a test method.

**Usage**

```
testSim(p = 0.99, r = 0.01, length = 400, A = 1, alpha = 1,
        beta = 0, phi = 0, frequency = 1, state = NULL, K,
        range = 200:400)
```

**Arguments**

p	probability to get a new epidemy at time i if there was one at time i-1, default 0.99
r	probability to get no new epidemy at time i if there was none at time i-1, default 0.01
length	number of weeks to model, default 400
A	amplitude (range of sinus), default = 1
alpha	parameter to move along the y-axis (negative values not allowed) with $\alpha > = A$ , default = 1
beta	regression coefficient, default = 0
phi	factor to create seasonal moves (moves the curve along the x-axis), default = 0
frequency	factor to determine the oscillation-frequency, default = 1
state	use a state chain to define the status at this timepoint (outbreak or not). If not given a Markov chain is generated by the programme, default NULL
K	additional weight for an outbreak which influences the distribution parameter $\mu$ , default = 0
range	range of timepoints to be evaluated by the RKI 1 system, default 200:400.

**Details**

A pointSource epidemy is generated and sent to the RKI 1 system, the quality values for the result are computed and shown as a latex table. Additionally a plot of the result is generated.

**Value**

xtable            one printed latex table and a result plot

**Author(s)**

M. Höhle, A. Riebler, C. Lang

**See Also**

[sim.pointSource](#), [algo.call](#), [algo.compare](#), [plot.survRes](#), [compMatrix.writeTable](#)

**Examples**

```
testSim(K = 2)
testSim(r = 0.5, K = 5)
```

---

toFileDisProg            *Writing of Disease Data*

---

**Description**

Writing of disease data (disProg object) into a file.

**Usage**

```
toFileDisProg(disProgObj, toFile)
```

**Arguments**

disProgObj            The disProgObj to save in file  
toFile                The path and filename of the file to save

**Details**

Writing of disProg object into a file as illustrated in the example.

**Value**

file                 The file with the disease data

**See Also**

[readData](#), [sim.pointSource](#)

**Examples**

```
disProgObj <- sim.pointSource(length=200, K=1)
toFileDisProg(disProgObj, "./simulation.txt")
mydisProgObj <- readData("./simulation",sysPath=FALSE)
```

**Description**

This function takes an `sts` object and applies an univariate surveillance algorithm to the time series of each observational unit.

**Usage**

```
wrap.algo(sts, algo, control, control.hook=function(k)
  return(control), verbose=TRUE, ...)

farrington(sts, control=list(range=NULL, b=3, w=3, reweight=TRUE,
  verbose=FALSE, alpha=0.01), ...)
cdc(sts, control=list(range=range, alpha=0.025), ...)
bayes(sts, control=list(range=range, b=0, w=6,
  actY=TRUE, alpha=0.05), ...)
rki(sts, control=list(range=range, b=2, w=4,
  actY=FALSE), ...)
cusum(sts, control=list(range=range, k=1.04, h=2.26,
  m=NULL, trans="standard", alpha=NULL), ...)
glrpois(sts, control=list(range=range, c.ARL=5, S=1, beta=NULL,
  Mtilde=1, M=-1, change="intercept", theta=NULL), ...)
glrnb(sts, control=list(range=range, c.ARL=5, mu0=NULL, alpha=0,
  Mtilde=1, M=-1, change="intercept",
  theta=NULL, dir=c("inc", "dec"),
  ret=c("cases", "value")), ...)
outbreakP(sts, control=list(range=range, k=100,
  ret=c("cases", "value"), maxUpperboundCases=1e5), ...)
```

**Arguments**

<code>sts</code>	Object of class <code>sts</code>
<code>algo</code>	Character string giving the function name of the algorithm to call, e.g. <code>"algo.farrington"</code> . Calling is done using <code>do.call</code> .
<code>control</code>	Control object as list. Depends on each algorithm.
<code>control.hook</code>	This is a function for handling multivariate objects. This argument is a function of integer <code>k</code> , which returns the appropriate control object for region <code>k</code>
<code>verbose</code>	Boolean, if <code>TRUE</code> then textual information about the process is given
<code>...</code>	Additional arguments sent to the <code>algo</code> function.

**Value**

An `sts` object with the `alarm`, `upperbound`, etc. slots filled with the results of independent and univariate surveillance algorithm.

**Author(s)**

M. Höhle

**See Also**

[algo.bayes](#), [algo.cdc](#), [algo.rki](#), [algo.farrington](#), [algo.cusum](#), [algo.glrpois](#), [algo.glrnb](#), [algo.outbreakP](#) for the exact form of the control object.

---

xtable.algoQV

*Xtable quality value object*


---

**Description**

Xtable a single quality value object in a nicely formatted way

**Usage**

```
## S3 method for class 'algoQV'
xtable(x,caption = NULL, label = NULL,
       align = NULL, digits = NULL, display = NULL, ...)
```

**Arguments**

x	Quality Values object generated with quality
caption	See <a href="#">xtable</a>
label	See <a href="#">xtable</a>
align	See <a href="#">xtable</a>
digits	See <a href="#">xtable</a>
display	See <a href="#">xtable</a>
...	Further arguments (see <a href="#">xtable</a> )

**See Also**

[xtable](#)

**Examples**

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

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

# Compute the quality values in a nice formatted way
```

```
library(xtable)
xtable(algo.quality(survResObj))
```

---

year-methods

*Methods for Function year*


---

### Description

Method to extract the corresponding year of each observation in an sts object. The year is calculated based on the start slot.

### Methods

`x = "sts"` For each time point in the x the corresponding year is determined and returned.

---

[-methods

*Methods for "[": Extraction or Subsetting in Package 'surveillance'*


---

### Description

Methods for "[", i.e., extraction or subsetting of the sts class in package **surveillance**.

Note that [<--methods methods (i.e. subassignments) are currently not supported.

drop is always FALSE.

### Methods

There are more than these:

```
x = "sts", i = "missing", j = "missing", drop = "ANY" ...
```

```
x = "sts", i = "numeric", j = "missing", drop = "missing" ...
```

```
x = "sts", i = "missing", j = "numeric", drop = "missing" ...
```

### Examples

```
data(ha)
#Convert to S4 object
shp <- system.file("shapes/berlin.shp", package="surveillance")
has4 <- aggregate(disProg2sts(ha, map=readShapePoly(shp, IDvar="SNAME")), nfreq=13)
```

```
#A suite of of simple tests (inspired by the Matrix package)
stopifnot(identical(has4, has4[]))
```

```
plot(has4[, 3])      # Single series
plot(has4[1:30, 3]) # Somewhat shorter
```

```
#Counts at time 20  
plot(has4[20, ],type = observed ~1 |unit)
```

# Index

- \*Topic **array**
  - [\[-methods, 105\]](#)
- \*Topic **classes**
  - [sts-class, 98](#)
- \*Topic **classif**
  - [algo.bayes, 7](#)
  - [algo.call, 9](#)
  - [algo.cdc, 10](#)
  - [algo.compare, 12](#)
  - [algo.cusum, 13](#)
  - [algo.farrington, 15](#)
  - [algo.glrnb, 19](#)
  - [algo.glrpois, 22](#)
  - [algo.hmm, 30](#)
  - [algo.outbreakP, 32](#)
  - [algo.rki, 36](#)
  - [algo.rogerson, 37](#)
  - [isoWeekYear, 65](#)
  - [wrap.algo, 103](#)
- \*Topic **cluster**
  - [stcd, 96](#)
- \*Topic **datagen**
  - [sim.pointSource, 92](#)
  - [sim.seasonalNoise, 93](#)
  - [simHHH, 94](#)
- \*Topic **datasets**
  - [abattoir, 5](#)
  - [deleval, 53](#)
  - [ha, 63](#)
  - [hepatitisA, 64](#)
  - [influMen, 64](#)
  - [m1, 69](#)
  - [measles.weser, 75](#)
  - [meningo.age, 75](#)
  - [momo, 76](#)
  - [salmonella.agona, 90](#)
  - [shadar, 91](#)
- \*Topic **data**
  - [CIdata, 48](#)
- \*Topic **file**
  - [toFileDisProg, 102](#)
- \*Topic **hplot**
  - [aggregate.disProg, 6](#)
  - [bestCombination, 44](#)
  - [create.disProg, 50](#)
  - [magic.dim, 70](#)
  - [plot.disProg, 82](#)
  - [plot.survRes, 84](#)
  - [primeFactors, 87](#)
  - [sumNeighbours, 100](#)
- \*Topic **methods**
  - [\[-methods, 105\]](#)
  - [aggregate-methods, 6](#)
  - [observed-methods, 77](#)
  - [obsinyear-methods, 77](#)
  - [year-methods, 105](#)
- \*Topic **misc**
  - [algo.hhh.grid, 27](#)
  - [algo.quality, 34](#)
  - [create.grid, 51](#)
  - [make.design, 71](#)
  - [makePlot, 72](#)
  - [readData, 88](#)
  - [test, 100](#)
  - [testSim, 101](#)
- \*Topic **models**
  - [arlCusum, 43](#)
  - [find.kh, 60](#)
  - [findH, 61](#)
  - [findK, 62](#)
  - [loglikelihood, 66](#)
  - [meanResponse, 73](#)
  - [predict.ah, 86](#)
  - [residuals.ah, 90](#)
- \*Topic **package**
  - [surveillance-package, 3](#)
- \*Topic **print**
  - [algo.summary, 39](#)

- compMatrix.writeTable, 48
- print.algoQV, 87
- xtable.algoQV, 104
- \*Topic **regression**
  - algo.farrington.assign.weights, 17
  - algo.farrington.fitGLM, 18
  - algo.farrington.threshold, 19
  - algo.hhh, 24
  - algo.twins, 41
  - anscombe.residuals, 43
  - categoricalCUSUM, 45
  - estimateGLRNbHook, 58
  - estimateGLRPoisHook, 59
  - LRCUSUM.runlength, 66
  - pairedbinCUSUM, 78
  - plot.atwins, 81
  - refvalIdxByDate, 89
- \*Topic **ts**
  - algo.hhh, 24
  - algo.twins, 41
  - display-methods, 54
  - plot.atwins, 81
- \*Topic **utilities**
  - correct53to52, 49
  - disProg2sts, 56
  - enlargeData, 57
- [, sts, ANY, ANY, ANY-method ([-methods), 105
- [, sts-method ([-methods), 105
- [-methods, 105
- abattoir, 5
- aggregate, 6
- aggregate, sts, ANY, ANY-method (aggregate-methods), 6
- aggregate, sts-method, 99
- aggregate, sts-method (aggregate-methods), 6
- aggregate-methods, 6
- aggregate.disProg, 6
- aggregate.ts, 6
- alarms (observed-methods), 77
- alarms, sts-method (sts-class), 98
- alarms-methods (observed-methods), 77
- alarms<- (observed-methods), 77
- alarms<- , sts-method (sts-class), 98
- algo.bayes, 7, 9, 11, 37, 104
- algo.bayes1 (algo.bayes), 7
- algo.bayes2 (algo.bayes), 7
- algo.bayes3 (algo.bayes), 7
- algo.bayesLatestTimepoint, 11, 37
- algo.bayesLatestTimepoint (algo.bayes), 7
- algo.call, 8, 9, 73, 102
- algo.cdc, 10, 104
- algo.cdcLatestTimepoint (algo.cdc), 10
- algo.compare, 12, 35, 40, 102
- algo.cusum, 13, 104
- algo.farrington, 9, 15, 18, 104
- algo.farrington.assign.weights, 17
- algo.farrington.fitGLM, 16, 18
- algo.farrington.threshold, 16, 19
- algo.glrnb, 19, 104
- algo.glrpois, 22, 58, 59, 104
- algo.hhh, 24, 29, 95
- algo.hhh.grid, 27, 52
- algo.hmm, 30
- algo.outbreakP, 32, 104
- algo.quality, 12, 34, 40
- algo.rki, 8, 9, 36, 48, 104
- algo.rki1 (algo.rki), 36
- algo.rki2 (algo.rki), 36
- algo.rki3 (algo.rki), 36
- algo.rkiLatestTimepoint, 8, 11, 21, 24
- algo.rkiLatestTimepoint (algo.rki), 36
- algo.rogerston, 37
- algo.summary, 39
- algo.twins, 41, 82
- anscombe.residuals, 17, 18, 43
- ar1Cusum, 43
- as.data.frame, sts-method (sts-class), 98
- bayes (wrap.algo), 103
- bestCombination, 44
- calc.outbreakP.statistic (algo.outbreakP), 32
- catcusum.LLRcompute (categoricalCUSUM), 45
- categoricalCUSUM, 5, 45, 46, 68, 79
- cdc (wrap.algo), 103
- CIdata, 48
- coef.ah (algo.hhh), 24
- coef.ahg (algo.hhh.grid), 27
- colnames, 99
- colnames, sts, missing, missing-method (sts-class), 98
- compMatrix.writeTable, 48, 102

- control (observed-methods), 77
- control, sts-method (sts-class), 98
- control-methods (observed-methods), 77
- control<- (observed-methods), 77
- control<- , sts-method (sts-class), 98
- correct53to52, 49, 73
- create.disProg, 50
- create.grid, 29, 51
- cusum (wrap.algo), 103
  
- delevel, 53
- dim, 99
- dim, sts-method (sts-class), 98
- display-methods, 54
- disProg2sts, 56
  
- enlargeData, 57, 73
- epoch (observed-methods), 77
- epoch, sts-method (sts-class), 98
- epoch-methods (observed-methods), 77
- epoch<- (observed-methods), 77
- epoch<- , sts-method (sts-class), 98
- epochInYear (observed-methods), 77
- epochInYear, sts-method (sts-class), 98
- epochInYear-methods (observed-methods), 77
- epochInYear<- (observed-methods), 77
- estimateGLRNbHook, 58
- estimateGLRPoisHook, 59
  
- farrington (wrap.algo), 103
- find.kh, 60
- findH, 38, 61
- findK, 62
  
- glrnb (wrap.algo), 103
- glrpois (wrap.algo), 103
  
- h1\_nrwrp, 88
- h1\_nrwrp (m1), 69
- ha, 63
- hepatitisA, 64
- hmm (wrap.algo), 103
- hValues, 39
- hValues (findH), 61
  
- influenMen, 64
- initialize, sts-method (sts-class), 98
- isoWeekYear, 65
  
- k1, 88
- k1 (m1), 69
  
- legend, 83, 85
- LLR.fun (LRCUSUM.runlength), 66
- loglikelihood, 66
- LRCUSUM.runlength, 66
  
- m1, 69, 88
- m2, 88
- m2 (m1), 69
- m3, 88
- m3 (m1), 69
- m4, 88
- m4 (m1), 69
- m5, 88
- m5 (m1), 69
- magic.dim, 70
- make.design, 71
- makePlot, 72
- meanResponse, 25, 26, 28, 29, 66, 71, 73
- measles.weser, 75
- meningo.age, 75
- momo, 76
- msm, 32
  
- n1, 88
- n1 (m1), 69
- n2, 88
- n2 (m1), 69
- ncol, sts-method (sts-class), 98
- nrow, sts-method (sts-class), 98
  
- observed (observed-methods), 77
- observed, sts-method (sts-class), 98
- observed-methods, 77
- observed<- (observed-methods), 77
- observed<- , sts-method (sts-class), 98
- obsinyear (obsinyear-methods), 77
- obsinyear, sts-method (obsinyear-methods), 77
- obsinyear-methods, 77
- optim, 61
- outbreakP (wrap.algo), 103
- outcomeFunStandard (LRCUSUM.runlength), 66
  
- pairedbinCUSUM, 53, 78
- plot, 55

- plot (display-methods), 54
- plot, sts, missing-method
  - (display-methods), 54
- plot.atwins, 81
- plot.disProg, 82, 99
- plot.sts.alarm (display-methods), 54
- plot.sts.spacetime (display-methods), 54
- plot.sts.time (display-methods), 54
- plot.survRes, 54, 55, 73, 84, 99, 102
- population (observed-methods), 77
- population, sts-method (sts-class), 98
- population-methods (observed-methods), 77
- population<- (observed-methods), 77
- population<-, sts-method (sts-class), 98
- predict.ah, 86
- predict.ahg (predict.ah), 86
- primeFactors, 87
- print.ah (algo.hhh), 24
- print.ahg (algo.hhh.grid), 27
- print.algoQV, 87
- print.default, 55
  
- q1\_nrwh, 88
- q1\_nrwh (m1), 69
- q2, 88
- q2 (m1), 69
  
- readData, 50, 57, 70, 73, 88, 102
- refvalIdxByDate, 89
- residuals.ah, 90
- residuals.ahg (residuals.ah), 90
- rki (wrap.algo), 103
- rogerson (wrap.algo), 103
  
- s1, 88
- s1 (m1), 69
- s2, 88
- s2 (m1), 69
- s3, 88
- s3 (m1), 69
- salmonella.agona, 90
- shadar, 91
- show, sts-method, 99
- show, sts-method (display-methods), 54
- sim.pointSource, 92, 94, 102
- sim.seasonalNoise, 92, 93
- simHHH, 94
- stcd, 96
  
- sts-class, 56
- sts-class, 98
- sts2disProg (disProg2sts), 56
- sumNeighbours, 100
- surveillance (surveillance-package), 3
- surveillance-package, 3
  
- test, 100
- testSim, 101
- toFileDisProg, 102
  
- upperbound (observed-methods), 77
- upperbound, sts-method (sts-class), 98
- upperbound<- (observed-methods), 77
- upperbound<-, sts-method (sts-class), 98
  
- wrap.algo, 103
  
- xtable, 104
- xtable.algoQV, 104
  
- year (year-methods), 105
- year, sts-method (year-methods), 105
- year-methods, 105