Appendix: The package surveillance

July 18, 2005

Title Test-bench for outbreak detection algorithms in surveillance data

Version 0.9

Author Höhle, Lang, Riebler

Description A framework for the development and the evaluation of outbreak detection algorithms in in routine collected public health surveillance data. Currently the package contains an implementation of the procedures described in Stroup et. al (1989), Farrington et. al (1996), a Bayesian approach and the method used at the Robert Koch Institute, Germany. The package contains several real-world datasets and the ability to simulate outbreak data.

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

License GPL version 2 (http://www.gnu.org/licenses/gpl.html)

URL http://www.stat.uni-muenchen.de/~hoehle/software/surveillance

R topics documented:

CIdata . . . 13
algo.bayes . . . 13
algo.call . . . 15
algo.cdc . . . 16
algo.compare . . . 18
algo.farrington . . . 19
algo.farrington.assign.weights . . . 20
algo.farrington.fitGLM . . . 21
algo.farrington.threshold . . . 22
algo.quality . . . 22
algo.rki . . . 23
algo.summary . . . 25
anscombe.residuals . . . 26
campylobacter . . . 27
compMatrix.writeTable . . . 27
correct53to52 . . . 28
enlargeData . . . 29
m1 . . . 30
makePlot . . . 31
plot.disProg . . . 32
plot.survRes . . . 33

print.algoQV 34
readData 35
salmonella.agona 36
sim.pointSource 36
sim.seasonalNoise 38
test 39
testSim 39
toFileDisProg 41
xtable.algoQV 41

Index **43**

`CIdata` *Confidence-Interval for the Mean of the Poisson Distribution*

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)
```

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 shoud be evaluated in algo.rkiLatestTimepoint. The default is to use the latest timepoint.
control	control object: range determines the desired timepoints which should be evaluated, b describes the number of years to go back for the reference values, w is the half window width for the reference values around the appropriate timepoint and actY is a boolean to decide if the year of timePoint also spend w reference values of the past. 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.

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

Examples

```
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 for bayes 1 the latest timepoint
algo.bayesLatestTimepoint(disProgObj)

# Test week 200 to 208 for outbreaks with a selfdefined bayes
algo.bayes(disProgObj, control = list(range = 200:208, b = 1,
                                      w = 5, actY = TRUE,alpha=0.05))
# The same for bayes 1 to bayes 3
algo.bayes1(disProgObj, control = list(range = 200:208,alpha=0.05))
algo.bayes2(disProgObj, control = list(range = 200:208,alpha=0.05))
algo.bayes3(disProgObj, 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 apropriate 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
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)
                        ) )
# this are some survResObjects
survRes$rki1
survRes$myBayes
```

---

algo.cdc *The CDC Algorithm*

Description

Surveillance using the CDC Algorithm

Usage

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

Arguments

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

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

which corresponds to Equation 8-1 in the Farrington and Andrews chapter. Note that an aggregation into 4-week blocks occurs and m denotes number of 4-week blocks (months) to use as reference values.

Value

survRes	algo.cdcLatestTimepoint returns a list of class survRes (surveillance result), which includes the alarm value (alarm = 1, no alarm = 0) for recognizing an outbreak, the threshold value for recognizing the alarm and the input object of class disProg. algo.cdc gives a list of class survRes which includes the vector of alarm values for every timepoint in range, the vector of threshold values for every timepoint in range for the system specified by b, w, the range and the input object of class disProg.

Author(s)

M. Höhle

Source

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

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

See Also

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

Examples

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

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

---

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

Description

Comparison of specified surveillance 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.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))
```

Arguments

`disProgObj`	object of class disProgObj (including the observed and the state chain)
`control`	Control object
range	Specifies the index of all timepoints which should be tested. If range is `NULL` the maximum number of possible weeks is used.
b	how many years back in time to include when forming the base counts.
w	windows size, i.e. number of weeks to include before and after the current week
reweight	Boolean specifying whether to perform reweight step
verbose	show extra debugging information
alpha	An approximate (two-sided) $(1-\alpha)\%$ confidence interval is calculated

Details

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

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

Value

An object of class `SurvRes`

Author(s)

M. Höhle

Source

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

See Also

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

Examples

```
#Read Salmonella Agona data
library(xtable)
salmonella.agona <- readData("salmonella.agona",week53to52=FALSE)

#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")
```

`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)
```

Arguments

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

Details

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

Value

An object of class GLM with additional fields `wtime`, `response` and `phi`

See Also

`anscombe.residuals`

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 <- function(pred,phi,alpha=0.01,
                                       skewness.transform="none")
```

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.
skeness.transform	Skewness correction, i.e. one of "none", "sqrt", or "2/3".

Value

vector	Vector of length 2 with lower and upper bounds of an $(1-\alpha)\%$ confidence interval.

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

Description

Computation of the quality values for a surveillance System output.

Usage

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

Arguments

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

Details

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

Value

`list of quality values`

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

See Also

`algo.compare`

Examples

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

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

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

`algo.rki` *The RKI System*

Description

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

Usage

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

Arguments

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

Details

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

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

Value

`survRes`	`algo.rkiLatestTimepoint` returns a list of class `survRes` (surveillance result), which includes the alarm value (alarm = 1, no alarm = 0) for recognizing an outbreak, the threshold value for recognizing the alarm and the input object of class disProg. `algo.rki` gives a list of class `survRes` which includes the vector of alarm values for every timepoint in `range`, the vector of threshold values for every timepoint in `range` for the system specified by `b`, `w` and `actY`, the range and the input object of class disProg. `algo.rki1` returns the same for the RKI 1 system, `algo.rki2` for the RKI 2 system and `algo.rki3` for the RKI 3 system.

Author(s)

M. Höhle, A. Riebler, Christian Lang

See Also

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

Examples

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

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

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

---

algo.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) )
```

---

anscombe.residuals *Compute Anscombe residuals*

Description

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

Usage

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

Arguments

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

Value

Standardized Anscombe residuals of `m`

References

McCullagh & Nelder, Generalized Linear Models, 1989

campylobacter *Weekly Campylobacter Reports in Germany 2001 - (mid) 2003*

Description

Reported number of campylobacter cases for each "Kreis" in Germany over the period 1 January 2001 (IfSG) to June 2003. The data have been provided to us by the Robert Koch Institute (RKI), Berlin, Germany.

Usage

```
data(campylobacter)
```

Format

A data frame with 130 weeks (rows) for all 439 Kreise (cols). Each entry is the number of reports in the Kreis during that week. See kreise.txt for information about the 439 kreise.

1001 Digit code for the 1st kreis, i.e. SK Flensburg.

... ...

16077 Digit Code for the last kreis, i.e. LK Altenburger Land

Source

Data have kindly been provided to us by the Robert Koch-Institut, Germany, 2003.

Examples

```
data(campylobacter)
#Show the number of cases for entire Germany
plot(apply(campylobacter,MARGIN=1,sum),type="l",ylab="reports",xlab="week no")
```

compMatrix.writeTable *Latex Table Generation*

Description

generates a latex table

Usage

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

Arguments

compMatrix	Matrix which includes quality values for every surveillance system.

Value

xtable	Latex table of the entered matrix.

Author(s)

M. Höhle, A. Riebler, C. Lang

Examples

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

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

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

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

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

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

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

---

correct53to52 *Data Correction from 53 to 52 weeks*

Description

Correction of data from 53 to 52 weeks a year

Usage

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

Arguments

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

Details

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

Value

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

Author(s)

Michael Höhle <http://www.stat.uni-muenchen.de/~hoehle>, Andrea Riebler, Christian Lang

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
```

---

enlargeData *Data Enlargement*

Description

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

Usage

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

Arguments

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

Details

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

Value

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

Author(s)

Michael Höhle <http://www.stat.uni-muenchen.de/~hoehle>, Andrea Riebler, Christian Lang

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
```

`m1` *RKI SurvStat Data*

Description

14 datasets for different diseases beginning in 2001 to the 3rd Quarter of 2004 including their defined outbreaks.

- `m1` 'Masern' in the 'Landkreis Nordfriesland' (Germany, Schleswig-Holstein)
- `m2` 'Masern' in the 'Stadt- und Landkreis Coburg' (Germany, Bayern)
- `m3` 'Masern' in the 'Kreis Leer' (Germany, Niedersachsen)
- `m4` 'Masern' in the 'Stadt- und Landkreis Aachen' (Germany, Nordrhein-Westfalen)
- `m5` 'Masern' in the 'Stadt Verden' (Germany, Niedersachsen)
- `q1_nrwh` 'Q-Fieber' in the 'Hochsauerlandkreis' (Germany, Westfalen) and in the 'Landkreis Waldeck-Frankenberg' (Germany, Hessen)
- `q2` 'Q-Fieber' in 'München' (Germany, Bayern)
- `s1` 'Salmonella Oranienburg' in Germany
- `s2` 'Salmonella Agona' in 12 'Bundesländern' of Germany
- `s3` 'Salmonella Anatum' in Germany
- `k1` 'Kryptosporidiose' in Germany, 'Baden-Württemberg'
- `n1` 'Norovirus' in 'Stadtkreis Berlin Mitte' (Germany, Berlin)
- `n2` 'Norovirus' in 'Torgau-Oschatz' (Germany, Sachsen)
- `h1_nrwrp` 'Hepatitis A' in 'Oberbergischer Kreis, Olpe, Rhein-Sieg-kreis' (Germany, Nordrhein-Westfalen) and 'Siegenwittgenstein Altenkirchen' (Germany, Rheinland-Pfalz)

Usage

```
data(m1)
```

Format

A data frame with 212 observations on the following 3 variables.

week weeknumber

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

```
require(surveillance)
disProgObj <- readData("k1")
    disProgObj <- correct53to52(disProgObj)
    survResObj <- algo.rki1(disProgObj, control=list(range=27:192))
    plot(survResObj, "RKI 1", "k1", firstweek=27, startyear=2002)
```

---

makePlot *Plot Generation*

Description

Just a test method.

Usage

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

Arguments

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

Details

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

Author(s)

M. Höhle, A. Riebler, C. Lang

See Also

readData, correct53to52, enlargeData, algo.call, plot.survRes

Examples

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

---

plot.disProg *Plot Generation of the Observed and the defined Outbreak State of a Timeseries*

Description

Plotting of a disProg object.

Usage

```
plot.disProg(x, title = "", startyear = 2001, firstweek = 1, ...)
```

Arguments

x	Object of class disProg
title	Plot title
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 grounds)
...	further arguments for the function matplot

Value

a plot	showing the number of infected and the defined alarm status for a timeseries created by simulation or given in data.

Author(s)

M. Höhle, A. Riebler, 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,
        startyear = 1999, firstweek = 13)
plot(disProgObj, title = title,
        startyear = 1999, firstweek = 14)
```

`plot.survRes` *Plot Generation*

Description

Plotting of a survRes object.

Usage

```
plot.survRes(x, method="", disease="",
                    startyear = 2001, firstweek = 1, legend=TRUE, ...)
```

Arguments

`x`	Object of class `survRes`
`method`	Surveillance method to be used in title
`disease`	Name of disease in title
`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)
`legend`	Boolean indicating whether to add a legend
`...`	further arguments for the function `matplot`

Value

`a plot`	showing the number of infected, the threshold for recognizing an outbreak, the computed alarm status and the defined alarm status.

Author(s)

M. Höhle, A. Riebler, C. Lang

Examples

```
# Plotting of simulated data
disProgObj <- sim.pointSource(p = 0.99, r = 0.5, length = 400,
                              A = 1, alpha = 1, beta = 0, phi = 0,
                              frequency = 1, state = NULL, K = 5)
# evaluate the timepoints defined by range using RKI 1
control <- list(list(funcName = "rki1", range = 200:400))
survResults <- algo.call(disProgObj, control = control)
# plot the result
plot(survResults[[1]], "RKI 1", "Simulation")
plot(survResults[[1]], "RKI 1", "Simulation",
                           firstweek = 13, startyear = 2002)
plot(survResults[[1]], "RKI 1", "Simulation", firstweek = 14)
```

---

print.algoQV *Print quality value object*

Description

Print a single qualitity value object in a nicely formatted way

Usage

```
print.algoQV <- function(algoQVObj)
```

Arguments

algoQV	Quality Values object generated with `quality`

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.

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

Value

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

Author(s)

Michael Höhle, Andrea Riebler, Christian Lang

See Also

m1, m2, m3, m4, m5, q1_nrwh, q2, s1, s2, s3, k1, n1, n2, h1_nrwrp

Examples

```
readData("m5")
```

salmonella.agona *Salmonella Agona cases in the UK 1990-1995*

Description

Reported number of cases of the Salmonella Agona serovar in the UK 1990-1995. Note however that the counts do not correspond exactly to the ones used by Farrington et. al (1996).

Usage

```
data(salmonella.agona)
```

Format

A 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="")
```

sim.pointSource *Generation of Simulated Point Source Epidemy*

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

M. Höhle, A. Riebler, C. Lang

See Also

sim.seasonalNoise

Examples

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

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

`sim.seasonalNoise` *Generation of Background Noise for Simulated Timeserieses*

Description

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

Usage

```
sim.seasonalNoise(A = 1, alpha = 1, beta = 0, phi = 0,
                  length, frequency = 1, state = NULL, K = 0)
```

Arguments

`A`	amplitude (range of sinus), default = 1.
`alpha`	parameter to move along the y-axis (negative values not allowed) with alpha >= A, default = 1.
`beta`	regression coefficient, default = 0.
`phi`	factor to create seasonal moves (moves the curve along the x-axis), default = 0.
`length`	number of weeks to model.
`frequency`	factor to determine the oscillation-frequency, default = 1.
`state`	if a state chain is entered the outbreaks will be additional weighted by K.
`K`	additional weigth for an outbreak which influences the distribution parameter mu, default = 0.

Value

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

Author(s)

M. Höhle, A. Riebler, C. Lang

See Also

`sim.pointSource`

Examples

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

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

test — *Print xtable for several diseases and the summary*

Description

Just a test method

Usage

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

Arguments

data	vector of abbreviations for the diseases
range	timepoints to evaluate

Details

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

Value

xtable	printed latex tables

Author(s)

M. Höhle, A. Riebler, C. Lang

Examples

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

testSim — *Print xtable for a Simulated Disease and the Summary*

Description

Just a test method.

Usage

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

Arguments

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

Details

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

Value

xtable	one printed latex table and a result plot

Author(s)

M. Höhle, A. Riebler, C. Lang

See Also

sim.pointSource, algo.call, algo.compare, plot.survRes, compMatrix.writeTable

Examples

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

toFileDisProg *Writing of Disease Data*

Description

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

Usage

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

Arguments

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

Details

Value

file	The file with the disease data

Author(s)

Michael Höhle <http://www.stat.uni-muenchen.de/~hoehle>, Andrea Riebler, Christian Lang

See Also

readData, sim.pointSource

Examples

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

xtable.algoQV *Xtable quality value object*

Description

Xtable a single qualitity value object in a nicely formatted way

Usage

```
xtable.algoQV <- function(algoQVObj)
```

Arguments

algoQV	Quality Values object generated with quality

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))
```

Index

∗Topic **datasets**
salmonella.agona, 35

algo.bayes, 13, *15*, *17*, *24*
algo.bayes1 *(algo.bayes)*, 13
algo.bayes2 *(algo.bayes)*, 13
algo.bayes3 *(algo.bayes)*, 13
algo.bayesLatestTimepoint, *17*, *24*
algo.bayesLatestTimepoint *(algo.bayes)*, 13
algo.call, 14, *31*, *39*
algo.cdc, 16
algo.cdcLatestTimepoint *(algo.cdc)*, 16
algo.compare, 17, *22*, *25*, *39*
algo.farrington, *15*, 18
algo.farrington.assign.weights, 20
algo.farrington.fitGLM, *19*, 20
algo.farrington.threshold, *19*, 21
algo.quality, *17*, 22, *25*
algo.rki, *13–15*, 23
algo.rki1 *(algo.rki)*, 23
algo.rki2 *(algo.rki)*, 23
algo.rki3 *(algo.rki)*, 23
algo.rkiLatestTimepoint, *14*, *17*
algo.rkiLatestTimepoint *(algo.rki)*, 23
algo.summary, 24
anscombe.residuals, *20, 21*, 25

campylobacter, 26
CIdata, 12
compMatrix.writeTable, 27, *39*
correct53to52, 28, *31*

enlargeData, 29, *31*

h1_nrwrp, *35*
h1_nrwrp *(m1)*, 30

k1, *35*
k1 *(m1)*, 30

m1, 30, *35*
m2, *35*
m2 *(m1)*, 30
m3, *35*
m3 *(m1)*, 30
m4, *35*
m4 *(m1)*, 30
m5, *35*
m5 *(m1)*, 30
makePlot, 31

n1, *35*
n1 *(m1)*, 30
n2, *35*
n2 *(m1)*, 30

plot.disProg, 32
plot.survRes, *31*, 33, *39*
print.algoQV, 34

q1_nrwh, *35*
q1_nrwh *(m1)*, 30
q2, *35*
q2 *(m1)*, 30

readData, *28–31*, 34, *40*

s1, *35*
s1 *(m1)*, 30
s2, *35*
s2 *(m1)*, 30
s3, *35*
s3 *(m1)*, 30
salmonella.agona, 35
sim.pointSource, 36, *37*, *39, 40*
sim.seasonalNoise, *36*, 37

test, 38
testSim, 39
toFileDisProg, 40

xtable.algoQV, 41