The R-Package 'surveillance'

SFB 386 Discussion Paper 422

Michael Höhle* and Andrea Riebler
Department of Statistics
University of Munich
Germany

July 18, 2005

Abstract

This document gives an introduction to the R-Package 'surveillance' containing tools for outbreak detection in routinely collected surveillance data. The package contains an implementation of the procedures described by Stroup et al. (1989), Farrington et al. (1996) and the system used at the Robert Koch Institute, Germany. For evaluation purposes, the package contains example data sets and functionality to generate surveillance data by simulation. To compare the algorithms, benchmark numbers like sensitivity, specificity, and detection delay can be computed for a set of time series. Being an open-source package it should be easy to integrate new algorithms; as an example of this process, a simple Bayesian surveillance algorithm is described, implemented and evaluated.

Keywords: infectious disease, monitoring, aberrations, outbreak, time series of counts.

1 Introduction

Public health authorities have in an attempt to meet the threats of infectious diseases to society created comprehensive mechanisms for the collection of disease data. As a consequence, the abundance of data has demanded the development of automated algorithms for the detection of abnormalities. Typically, such an algorithm monitors a univariate time series of counts using a combination of heuristic methods and statistical modelling. Prominent examples of surveillance algorithms are the work by Stroup et al. (1989)

*Author of correspondance: Department of Statistics, University of Munich, Ludwigstr. 33, 80539 München, Germany, Email: hoehle@stat.uni-muenchen.de

and Farrington et al. (1996). A comprehensive survey of outbreak detection methods can be found in (Farrington and Andrews, 2003).

The R-package `surveillance` was written with the aim of providing a test-bench for surveillance algorithms. From

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

the package can be downloaded together with its source code. It allows users to test new algorithms and compare their results with those of standard surveillance methods. A few real world outbreak datasets are included together with mechanisms for simulating surveillance data. With the package at hand, comparisons like the one described by Hutwagner et al. (2005) should be easy to conduct.

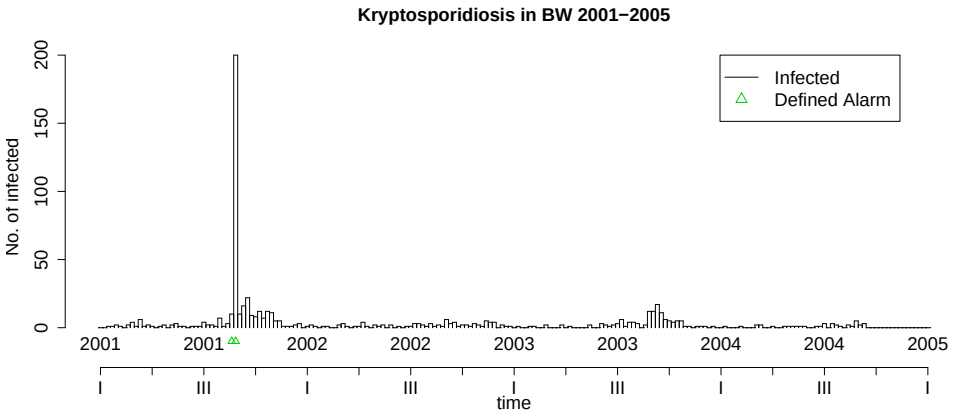
The purpose of this document is to illustrate the basic functionality of the package with R-code examples. Section 2 contains a description of the data format used to store surveillance data, mentions the built-in datasets and illustrates how to create new datasets by simulation. Section 3 contains a short description of how to use the surveillance algorithms and illustrate the results. Further information on the individual functions can be found in the on-line documentation of the package, which is also provided in printed form as an Appendix of this document.

2 Surveillance Data

Denote by $\{y_t\,; t = 1, \ldots, n\}$ the time series of counts representing the surveillance data. Because such data typically are collected on a weekly basis, we shall also use the alternative notation $\{y_{i:j}\}$ with $j = \{1, \ldots, 52\}$ being the week number in year $i = \{-b, \ldots, -1, 0\}$. That way the years are indexed such that most current year has index zero. For evaluation of the outbreak detection algorithms it is also possible for each week to store – if known – whether there was an outbreak that week. The resulting multivariate series $\{(y_t, x_t)\,; t = 1, \ldots, n\}$ is in `surveillance` given by an object of class `disProg` (disease progress), which is basically a `list` containing two vectors: the observed number of counts and a boolean vector `state` indicating whether there was an outbreak that week. A number of time series are contained in the `data` directory, mainly originating from the SurvStat@RKI database at http://www3.rki.de/SurvStat/ maintained by the Robert Koch Institute, Germany (Robert Koch-Institut, 2004). For example the object `k1` describes Kryptosporidosis surveillance data for the German federal state Baden-Württemberg 2001-2005. The peak in 2001 is due to an outbreak of Kryptosporidosis among a group of army-soldiers in boot-camp (Robert Koch Institute, 2001). In `surveillance` the `readData` function is used to bring the time series on `disProg` form. The SurvStat@RKI database at http://www3.rki.de/SurvStat/ maintained by the Robert Koch Institute,

Germany, uses a 53 weeks a year format; therefore a conversion with `correct53to52` is necessary.

```
> k1 <- readData("k1", week53to52 = TRUE)
> plot(k1, main = "Kryptosporidiosis in BW 2001-2005")
```



For evaluation purposes it is also of interest to generate surveillance data using simulation. The package contains functionality to generate surveillance data containing point-source like outbreaks, for example with a Salmonella serovar. The model is a Hidden Markov Model (HMM) where a binary state $X_t, t = 1, \ldots, n$, denotes whether there was an outbreak and Y_t is the number of observed counts, see Fig. 1.

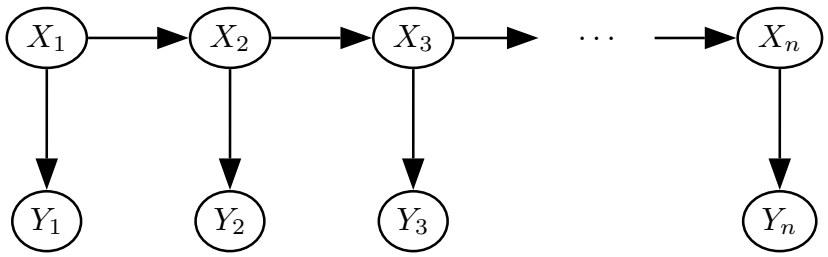


Figure 1: The Hidden Markov Model

The state X_t is a homogenous Markov chain with the following transition matrix

$X_t \backslash X_{t+1}$	0	1
0	p	$1-p$
1	$1-r$	r

Hence $1-p$ is the probability to switch to an outbreak state and $1-r$ is the probability that $X_t = 1$ is followed by $X_{t+1} = 1$. Furthermore, the

observation Y_t is Poisson-distributed with log-link mean depending on a seasonal effect and time trend, i.e.

$$\log \mu_t = A \cdot \sin(\omega \cdot (t + \varphi)) + \alpha + \beta t.$$

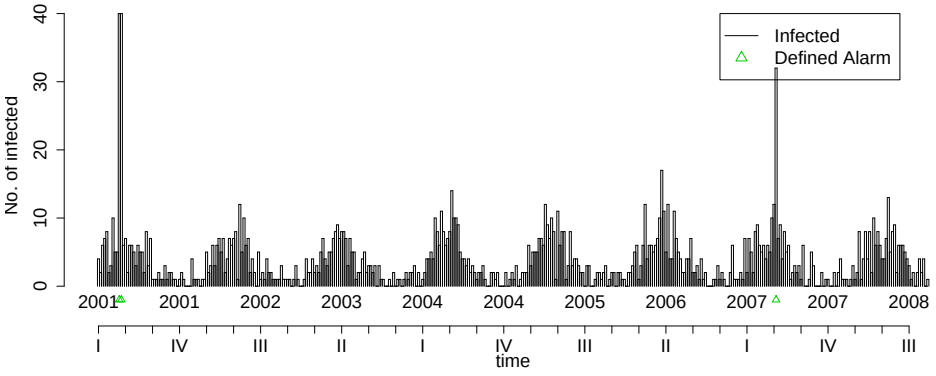
In case of an outbreak ($X_t = 1$) the mean increases with a value of K, altogether

$$Y_t \sim \text{Po}(\mu_t + K \cdot X_t). \tag{1}$$

The model in (1) corresponds to a single-source, common-vehicle outbreak, where the length of an outbreak is controlled by the transition probability r. The daily numbers of outbreak-cases are simply independently Poisson distributed with mean K. A physiologically better motivated alternative could be to operate with a stochastic incubation time (e.g. log-normal or gamma distributed) for each individual exposed to the source, which results in a temporal diffusion of the peak. The advantage of (1) is that estimation can be done by a generalized linear model (GLM) using X_t as covariate and that it allows for an easy definition of a correctly identified outbreak: each $X_t = 1$ has to be identified. More advanced setups would require more involved definitions of an outbreak, e.g. as a connected series of time instances, where the number of outbreak cases is greater than zero. Care is then required in defining what a correctly identified outbreak for time-wise overlapping outbreaks means.

In `surveillance` the function `sim.pointSource` is used to simulate such a point-source epidemic; the result is an object of class `disProg`.

```
> sts <- 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)
> plot(sts)
```



3 Surveillance Algorithms

Surveillance data often exhibit strong seasonality, therefore most surveillance algorithms only use a set of so called *reference values* as basis for drawing conclusions. Let $y_{0:t}$ be the number of cases of the current week (denoted week t in year 0), b the number of years to go back in time and w the number of weeks around t to include from those previous years. For the year zero we use w_0 as the number of previous weeks to include – typically $w_0 = w$. Altogether the set of reference values is thus defined to be

$$R(w, w_0, b) = \left(\bigcup_{i=1}^{b} \bigcup_{j=-w}^{w} y_{-i:t+j} \right) \cup \left(\bigcup_{k=-w_0}^{-1} y_{0:t+k} \right)$$

Note that the number of cases of the current week is not part of $R(w, w_0, b)$.

A surveillance algorithm is a procedure using the reference values to create a prediction $\hat{y}_{0:t}$ for the current week. This prediction is then compared with the observed $y_{0:t}$: if the observed number of cases is much higher than the predicted number, the current week is flagged for further investigations. In order to do surveillance for time $0 : t$ an important concern is the choice of b and w. Values as far back as time $-b : t - w$ contribute to $R(w, w_0, b)$ and thus have to exist in the observed time series.

Currently, we have implemented four different type of algorithms in `surveillance`. The Centers for Disease Control and Prevention (CDC) method (Stroup et al., 1989), the Communicable Disease Surveillance Centre (CDSC) method (Farrington et al., 1996), the method used at the Robert Koch Institute (RKI), Germany (Altmann, 2003), and a Bayesian approach documented in Riebler (2004). A detailed description of each method is beyond the scope of this note, but to give an idea of the framework the Bayesian approach developed in Riebler (2004) is presented: Within a Bayesian framework, quantiles of the predictive posterior distribution are used as a measure for defining alarm thresholds.

The model assumes that the reference values are identically and independently Poisson distributed with parameter λ and a Gamma-distribution is used as Prior distribution for λ. The reference values are defined to be $R_{\text{Bayes}} = R(w, w_0, b) = \{y_1, \ldots, y_n\}$ and $y_{0:t}$ is the value we are trying to predict. Thus, $\lambda \sim \text{Ga}(\alpha, \beta)$ and $y_i|\lambda \sim \text{Po}(\lambda)$, $i = 1, \ldots, n$. Standard derivations show that the posterior distribution is

$$\lambda|y_1, \ldots, y_n \sim \text{Ga}(\alpha + \sum_{i=1}^{n} y_i, \beta + n).$$

Computing the predictive distribution

$$f(y_{0:t}|y_1, \ldots, y_n) = \int_0^\infty f(y_{0:t}|\lambda)\, f(\lambda|y_1, \ldots, y_n)\, d\lambda$$

we get the Poisson-Gamma-distribution

$$y_{0:t}|y_1,\ldots,y_n \sim \text{PoGa}(\alpha+\sum_{i=1}^{n} y_i, \beta+n),$$

which is a generalization of the negative Binomial distribution, i.e.

$$y_{0:t}|y_1,\ldots,y_n \sim \text{NegBin}(\alpha+\sum_{i=1}^{n} y_i, \tfrac{\beta+n}{\beta+n+1}).$$

Using the Jeffrey's Prior $\text{Ga}(\frac{1}{2},0)$ as non-informative Prior distribution for λ the parameters of the negative Binomial distribution are

$$\alpha+\sum_{i=1}^{n} y_i = \frac{1}{2}+\sum_{y_{i:j}\in R_{\text{Bayes}}} y_{i:j} \qquad \text{and} \qquad \frac{\beta+n}{\beta+n+1} = \frac{|R_{\text{Bayes}}|}{|R_{\text{Bayes}}|+1}.$$

Using a quantile-parameter α, the smallest value y_α is computed, so that

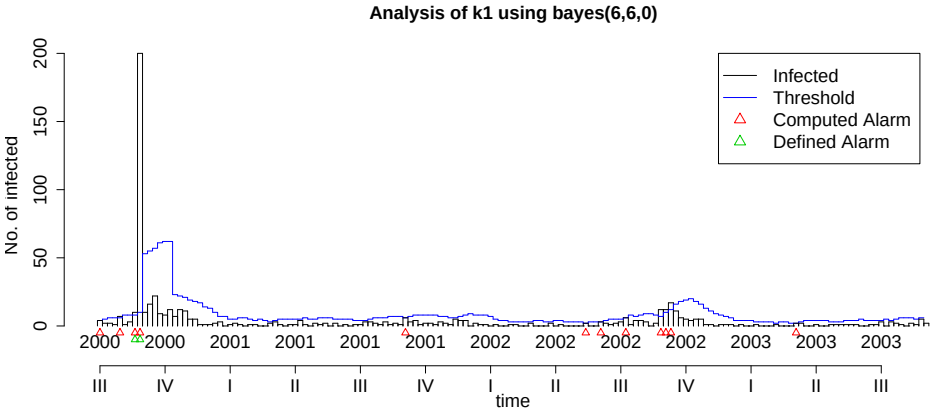
$$P(y \le y_\alpha) \ge 1-\alpha.$$

Now

$$A_{0:t} = I(y_{0:t} \ge y_\alpha),$$

i.e. if $y_{0:t} \ge y_\alpha$ the current week is flagged as an alarm. As an example, the `Bayes1` method uses the last six weeks as reference values, i.e. $R(w,w_0,b) = (6,6,0)$, and is applied to the `k1` dataset with $\alpha = 0.01$ as follows.

```
> k1.b660 <- algo.bayes(k1, control = list(range = 27:192,
+     b = 0, w = 6, alpha = 0.01))
> plot(k1.b660, disease = "k1", firstweek = 1, startyear = 2001)
```



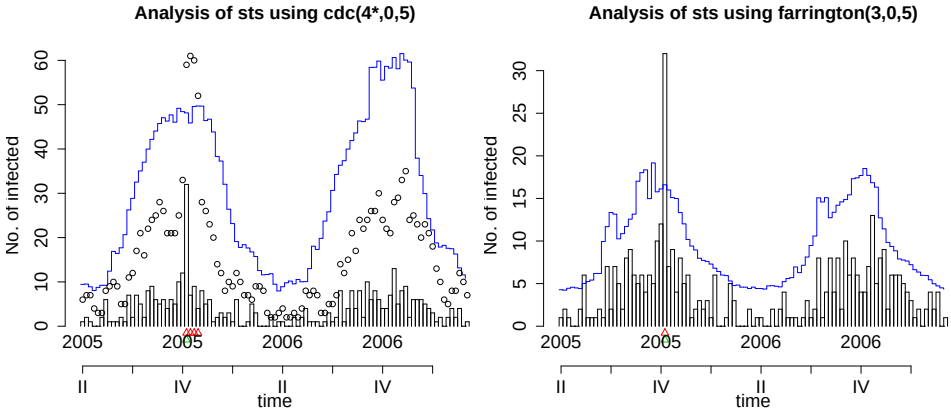
Several extensions of this simple Bayesian approach are imaginable, for example the inane over-dispersion of the data could be modeled by using

a negative-binomial distribution, time trends and mechanisms to correct for past outbreaks could be integrated, but all at the cost of non-standard inference for the predictive distribution. Here simulation based methods like Markov Chain Monte Carlo or heuristic approximations have to be used to obtain the required alarm thresholds.

In general, the `surveillance` package makes it easy to add additional algorithms – also those not based on reference values – by using the existing implementations as starting point.

The following call uses the CDC and Farrington procedure on the simulated time series `sts` from page 4. Note that the CDC procedure operates with four-week aggregated data – to better compare the upper bound value, the aggregated number of counts for each week are shown as circles in the plot.

```
> par(mfcol = c(1, 2))
> cntrl <- list(range = 300:400, m = 1, w = 3, b = 5, alpha = 0.01)
> sts.cdc <- algo.cdc(sts, control = cntrl)
> sts.farrington <- algo.farrington(sts, control = cntrl)
> plot(sts.cdc, legend = F)
> plot(sts.farrington, legend = F)
```



Typically, one is interested in evaluating the performance of the various surveillance algorithms. An easy way is to look at the sensitivity and specificity of the procedure – a correct identification of an outbreak is defined as follows: if the algorithm raises an alarm for time t, i.e. $A_t = 1$ and $X_t = 1$ we have a correct classification, if $A_t = 1$ and $X_t = 0$ we have a false-positive, etc. In case of more involved outbreak models, where an outbreak lasts for more than one week, a correct identification could be if at least one of the outbreak weeks is correctly identified, see e.g. Hutwagner et al. (2005).

To compute various performance scores the function `algo.quality` can be used on a `SurvRes` object.

```
> print(algo.quality(k1.b660))
```

```
     TP FP TN  FN Sens Spec      dist       mlag
[1,] 2  10 154 0  1    0.9390244 0.06097561 0
```

This computes the number of false positives, true negatives, false negatives, the sensitivity and the specificity. Furthermore, `dist` is defined as

$$\sqrt{(Spec-1)^2+(Sens-1)^2},$$

that is the distance to the optimal point $(1,1)$, which serves as a heuristic way of combining sensitivity and specificity into a single score. Of course, weighted versions are also imaginable. Finally, `lag` is the average number of weeks between the first of a consecutive number of $X_t = 1$'s (i.e. an outbreak) and the first alarm raised by the algorithm.

To compare the results of several algorithms on a single time series we declare a list of control objects – each containing the name and settings of the algorithm we want to apply to the data.

```
> control = list(list(funcName = "rki1"), list(funcName = "rki2"),
+     list(funcName = "rki3"), list(funcName = "bayes1"),
+     list(funcName = "bayes2"), list(funcName = "bayes3"),
+     list(funcName = "cdc", alpha = 0.05), list(funcName = "farrington",
+         alpha = 0.05))
> control <- lapply(control, function(ctrl) {
+     ctrl$range <- 300:400
+     return(ctrl)
+ })
```

In the above, `rki1`, `rki2` and `rki3` are three methods with reference values $R_{\text{rki1}}(6,6,0)$, $R_{\text{rki2}}(6,6,1)$ and $R_{\text{rki3}}(4,0,2)$ all called with $\alpha = 0.05$. The methods `bayes1-bayes3` is the Bayesian algorithm using the same setup of reference values. The CDC Method is special, since it operates on aggregated four-week blocks. To make everything comparable a common $\alpha = 0.05$ level is used for all algorithms. All algorithms in `control` are applied to `sts` using:

```
> algo.compare(algo.call(sts, control = control))
```

```
                  TP FP TN FN sens spec dist mlag
rki(6,6,0)        1  7  93 0  1    0.93 0.07 0
rki(6,6,1)        1  1  99 0  1    0.99 0.01 0
rki(4,0,2)        1  1  99 0  1    0.99 0.01 0
bayes(6,6,0)      1  13 87 0  1    0.87 0.13 0
bayes(6,6,1)      1  10 90 0  1    0.9  0.1  0
bayes(4,0,2)      1  7  93 0  1    0.93 0.07 0
cdc(4*,0,5)       1  4  96 0  1    0.96 0.04 0
farrington(3,0,5) 1  3  97 0  1    0.97 0.03 0
```

A test on a set of time series can be done as follows. Firstly, a list containing 10 simulated time series is created. Secondly, all the algorithms specified in the `control` object are applied to each series. Finally the results for the 10 series are combined in one result matrix.

```
> ten <- lapply(1:10, function(x) {
+     sim.pointSource(p = 0.975, r = 0.5, length = 400,
+         A = 1, alpha = 1, beta = 0, phi = 0, frequency = 1,
+         state = NULL, K = 1.7)
+ })
> ten.surv <- lapply(ten, function(ts) {
+     algo.compare(algo.call(ts, control = control))
+ })

> algo.summary(ten.surv)

                   TP FP  TN FN sens spec dist mlag
rki(6,6,0)         31 25 941 13 0.70 0.97 0.30 1.82
rki(6,6,1)         35 11 955  9 0.80 0.99 0.20 1.90
rki(4,0,2)         37  5 961  7 0.84 0.99 0.16 1.90
bayes(6,6,0)       35 86 880  9 0.80 0.91 0.22 0.67
bayes(6,6,1)       40 52 914  4 0.91 0.95 0.11 0.35
bayes(4,0,2)       42 46 920  2 0.95 0.95 0.07 1.00
cdc(4*,0,5)        23 30 936 21 0.52 0.97 0.48 8.90
farrington(3,0,5) 29 12 954 15 0.66 0.99 0.34 4.79
```

A similar procedure can be applied when evaluating the 14 surveillance series drawn from SurvStat@RKI (Robert Koch-Institut, 2004). A problem is however, that the series after conversion to 52 weeks/year are of length 209 weeks. This is insufficient to apply e.g. the CDC algorithm. To conduct the comparison on as large a dataset as possible the following trick is used: The function `enlargeData` replicates the requested `range` and inserts it before the original data, after which the evaluation can be done on all 209 values.

```
> range = (2 * 4 * 52) + 1:length(k1$observed)
> control <- lapply(control, function(cntrl) {
+     cntrl$range = range
+     return(cntrl)
+ })
> outbrks <- c("m1", "m2", "m3", "m4", "m5", "q1_nrwh",
+     "q2", "s1", "s2", "s3", "k1", "n1", "n2", "h1_nrwrp")
> outbrks <- lapply(outbrks, function(name) {
+     enlargeData(readData(name), range = 1:(4 * 52), times = 2)
+ })
> one.survstat.surv <- function(outbrk) {
+     algo.compare(algo.call(outbrk, control = control))
+ }
```

```
> algo.summary(lapply(outbrks, one.survstat.surv))

                    TP  FP   TN  FN sens spec dist mlag
rki(6,6,0)          38  62 2646 180 0.17 0.98 0.83 5.43
rki(6,6,1)          65  83 2625 153 0.30 0.97 0.70 5.57
rki(4,0,2)          80 106 2602 138 0.37 0.96 0.63 5.43
bayes(6,6,0)        61 206 2502 157 0.28 0.92 0.72 1.71
bayes(6,6,1)       123 968 1740  95 0.56 0.64 0.56 1.36
bayes(4,0,2)       162 920 1788  56 0.74 0.66 0.43 1.36
cdc(4*,0,5)         65  94 2614 153 0.30 0.97 0.70 7.14
farrington(3,0,5)   25  26 2682 193 0.11 0.99 0.89 8.21
```

In both this study and the earlier simulation study the Bayesian approach seems to do quite well. However, the extent of the comparisons do not make allowance for any more supported statements. Consult the work of Riebler (2004) for a more thorough comparision using simulation studies.

4 Discussion and Future work

Many extensions and additions are imaginable to improve the package. For now, the package is intended as an academic tool providing a test-bench for integrating new surveillance algorithms. Because all algorithms are implemented in R, performance has not been an issue. Especially the current implementation of the Farrington Procedure is rather slow and would benefit from an optimization possible with fragments written in C.

One important improvement would be to provide more involved mechanisms for the simulation of epidemics. In particular it would be interesting to include multi-day outbreaks originating from single-source exposure, but with delay due to varying incubation time (Hutwagner et al., 2005) or SEIR-like epidemics (Andersson and Britton, 2000). However, defining what is meant by a correct outbreak identification, especially in the case of overlapping outbreaks, creates new challenges which have to be met.

5 Acknowledgements

We are grateful to K. Stark and D. Altmann, RKI, Germany, for discussions and information on the surveillance methods used by the RKI. Our thanks to C. Lang, University of Munich, for his work on the R–implementation and M. Kobl, T. Schuster and M. Rossman, University of Munich, for their initial work on gathering the outbreak data from SurvStat@RKI. The research was conducted with financial support from the Collaborative Research Centre SFB 386 funded by the German research foundation (DFG).

References

Altmann, D. (2003). The Surveillance System of the Robert Koch Institute, Germany. Personal Communication.

Andersson, H. and T. Britton (2000). *Stochastic Epidemic Models and their Statistical Analysis*, Volume 151 of *Springer Lectures Notes in Statistics*. Springer-Verlag.

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.

Farrington, C., N. Andrews, A. Beale, and M. Catchpole (1996). A statistical algorithm for the early detection of outbreaks of infectious disease. *Journal of the Royal Statistical Society, Series A 159*, 547–563.

Hutwagner, L., T. Browne, G. Seeman, and A. Fleischhauer (2005). Comparing abberation detection methods with simulated data. *Emerging Infectious Diseases 11*, 314–316.

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

Robert Koch-Institut (2004). SurvStat@RKI. http://www3.rki.de/SurvStat. Date of query: September 2004.

Robert Koch Institute (2001). Epidemiologisches Bulletin 39. Available from http://www.rki.de.

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.

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` *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)
```