Package ‘STAR’ documentation

of

November 7, 2007

Type Package

Title Spike Train Analysis with R

Version 0.1-5

Date 2007-11-07

Depends survival, sound, mgcv, R2HTML

Suggests gam, lattice, ggplot2, HiddenMarkov

Author Christophe Pouzat

Maintainer Christophe Pouzat <christophe.pouzat@gmail.com>

Description Functions to analyze neuronal spike trains

License GPL version 2 or newer

URL http://www.biomedicale.univ-paris5.fr/physcerv/C_Pouzat/STAR.html

R topics documented:

`acf.spikeTrain` *Auto- Covariance and -Correlation Function Estimation for Spike Train ISIs*

Description

The function `acf.spikeTrain` computes (and by default plots) estimates of the autocovariance or autocorrelation function of the inter-spike intervals of a spike train.

Usage

```
acf.spikeTrain(spikeTrain, lag.max = NULL,
       type = c("correlation", "covariance", "partial"),
       plot = TRUE, na.action = na.fail,
       demean = TRUE, xlab = "Lag (in isi #)",
       ylab = "ISI acf",
       main, ...)
```

Arguments

`spikeTrain`	a `spikeTrain` object or a vector which can be coerced to such an object.
`lag.max`	maximum lag at which to calculate the acf. Default is $10 \log_{10}(N)$ where N is the number of ISIs. Will be automatically limited to one less than the number of ISIs in the spike train.
`type`	character string giving the type of acf to be computed. Allowed values are `"correlation"` (the default), `"covariance"` or `"partial"`.
`plot`	logical. If `TRUE` (the default) the acf is plotted.
`na.action`	function to be called to handle missing values. `na.pass` can be used.
`demean`	logical. Should the covariances be about the sample means?
`xlab`	x axis label.
`ylab`	y axis label.
`main`	title for the plot.
`...`	further arguments to be passed to `plot.acf`.

Details

Just a wrapper for `acf` function. The first argument, `spikeTrain`, is processed first to extract the inter-spike intervals. `acf.spikeTrain` is mainly used to plot what Perkel et al (1967) call the *serial correlation coefficient* (Eq. 8) or *serial covariance coefficient* (Eq. 7), p 400.

Value

An object of class `"acf"`, which is a list with the following elements:

`lag`	A three dimensional array containing the lags at which the acf is estimated.
`acf`	An array with the same dimensions as `lag` containing the estimated acf.
`type`	The type of correlation (same as the `type` argument).
`n.used`	The number of observations in the time series.
`series`	The name of the series `x`.
`snames`	The series names for a multivariate time series.

The lag `k` value returned by `ccf(x,y)` estimates the correlation between `x[t+k]` and `y[t]`.

The result is returned invisibly if `plot` is `TRUE`.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Perkel D. H., Gerstein, G. L. and Moore G. P. (1967) Neural Spike Trains and Stochastic Point Processes. I. The Single Spike Train. *Biophys. J.*, **7**: 391-418. http://www.pubmedcentral.nih.gov/articlerender.fcgi?tool=pubmed\&pubmedid=4292791

See Also

`acf`, `varianceTime`, `renewalTestPlot`

Examples

```
## Simulate a log normal train
train1 <- c(cumsum(rlnorm(301,log(0.01),0.25)))
train1 <- as.spikeTrain(train1)
## Get its isi acf
acf.spikeTrain(train1,lag.max=100)
```

`as.repeatedTrain` *Coerce and Test repeatedTrain Objects*

Description

`as.repeatedTrain` attempts to coerce a list with numeric vector elements to a `repeatedTrain` object while `is.repeatedTrain` tests if its argument is such an object.

Usage

```
as.repeatedTrain(x)
is.repeatedTrain(x)
```

Arguments

`x`	An object to be coerced to or to test against a `repeatedTrain` object.

Details

A `repeatedTrain` object is list of `spikeTrain` objects. It is used to store the responses of a given neuron to repeated stimulations.

Value

`as.repeatedTrain` returns a `repeatedTrain` object or an error.

`is.repeatedTrain` returns `TRUE` if its argument is a `repeatedTrain` object and `FALSE` otherwise.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`plot.repeatedTrain`, `print.repeatedTrain`, `summary.repeatedTrain`, `psth`, `raster`, `as.spikeTrain`, `is.spikeTrain`

Examples

```
## load CAL1V data
data(CAL1V)
## convert them to repeatedTrain objects
CAL1V <- lapply(CAL1V, as.repeatedTrain)
## did the conversion work?
sapply(CAL1V, is.repeatedTrain)
## look at the raster of the 1st neuron
CAL1V[["neuron 1"]]
```

`as.spikeTrain` *Coerce, Test and Extract from spikeTrain Objects*

Description

`as.spikeTrain` attempts to coerce a numeric vector to a `spikeTrain` object while `is.spikeTrain` tests if its argument is such an object. `[.spikeTrain`, extracts a subset of a `spikeTrain` object.

Usage

```
as.spikeTrain(x)
is.spikeTrain(x)
## S3 method for class 'spikeTrain':
x[i]
```

Arguments

`x`	An object to be coerced to or to test against a `spikeTrain` object or a `spikeTrain` obect for `[`.
`i`	indices specifying elements to extract. *No gaps are allowed.*

Details

A `spikeTrain` object is a `numeric` vector whose elements are strictly increasing (that is, something which can be interpreted as a sequence of times of successive events with no two events occurring at the same time). The extractor method, `[` requires that the extracted elements are without gaps, an error is returned otherwise.

Value

`as.spikeTrain` returns a `spikeTrain` object or an error.

`is.spikeTrain` returns `TRUE` if its argument is a `spikeTrain` object and `FALSE` otherwise.

`[` returns a `spikeTrain` object or an error.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Perkel D. H., Gerstein, G. L. and Moore G. P. (1967) Neural Spike Trains and Stochastic Point Processes. I. The Single Spike Train. *Biophys. J.*, **7**: 391-418. http://www.pubmedcentral.nih.gov/articlerender.fcgi?tool=pubmed\&pubmedid=4292791

See Also

`plot.spikeTrain`, `print.spikeTrain`, `summary.spikeTrain`

Examples

```
## load CAL1S data
data(CAL1S)
## convert the data into spikeTrain objects
CAL1S <- lapply(CAL1S,as.spikeTrain)
## Are the list eleemnts now spikeTrain objects?
sapply(CAL1S, is.spikeTrain)
## look at the train of the 1st neuron
CAL1S[["neuron 1"]]
## look at the window 10-40 using the extractor function
CAL1S[["neuron 1"]][10 < CAL1S[["neuron 1"]] & CAL1S[["neuron 1"]] < 40]
```

`brt4df` *Get Backward Recurrence Times from Data Frames Generated by mkGLMdf*

Description

Spike trains discharge models for single neurons are rarely renewal. They require more information than just the elapsed time since the last spike. Function `brt4df` generates this additional information from a data frame obtained by `mkGLMdf`.

Usage

```
brt4df(df, varName, max.order = 1, colNames,
       auto = TRUE, normalise = function(x) as.numeric(scale(log(x))))
```

Arguments

`df`	A `data.frame` generated by `mkGLMdf` and containing the events of a single neuron.
`varName`	The name of one of the variables of `df`. It should be one of the "elapsed time" variables, like, `lN.x`, where `x` stands for a neuron number.
`max.order`	How many events should looked for in the past?
`colNames`	Names of the columns of the returned `data.frame`. If missing default names are provided.
`auto`	A logical. Does `varName` refer to the elapsed times since the last spike of the neuron whose spikes are recorded in the `event` variable (`TRUE`) or not (`FALSE`)?
`normalise`	A `function` applied to the extracted data in order to normalise them. If missing ,nothing is done and the extracted data are left unchanged.

Details

If the spike required to evaluate the elapsed time is not contained in df then NA will be the reported elapsed time.

Value

A data.frame is returned with as many variable as max.order and as many rows as df.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Kass, Robert E. and Ventura, Valérie (2001) A spike-train probability model *Neural Comput.* **13**: 1713–1720.

Truccolo, W., Eden, U. T., Fellows, M. R., Donoghue, J. P. and Brown, E. N. (2005) A Point Process Framework for Relating Neural Spiking Activity to Spiking History, Neural Ensemble and Extrinsic Covariate Effects *J Neurophysiol* **93**: 1074–1089. http://jn.physiology.org/cgi/content/abstract/93/2/1074

See Also

mkGLMdf, data.frame, glm, mgcv

Examples

```
## Not run:
## Let us consider neuron 1 of the CAL2S data set
data(CAL2S)
CAL2S <- lapply(CAL2S,as.spikeTrain)
CAL2S[["neuron 1"]]
renewalTestPlot(CAL2S[["neuron 1"]])
summary(CAL2S[["neuron 1"]])
## Make a data frame with a 4 ms time resolution
cal2Sdf <- mkGLMdf(CAL2S,0.004,0,60)
## keep the part relative to neuron 1
n1.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="1",]
## remove unnecessary data
rm(cal2Sdf)
## Extract the elapsed time since the second to last and
## third to last for neuron 1. Normalise the result.
n1.cal2sDF[c("rlN.1","rsN.1","rtN.1")] <- brt4df(n1.cal2sDF,"lN.1",2,c("rlN.1","rsN.1","r
## load mgcv library
library(mgcv)
## fit a model with a tensorial product involving the last
## three spikes and using a cubic spline basis for the last two
n1S.fitA <- gam(event ~ te(rlN.1,rsN.1,bs="cr") + rtN.1,data=n1.cal2sDF,family=binomial(l
summary(n1S.fitA)
## plot the result in 2 different ways
plot(n1S.fitA)
vis.gam(n1S.fitA,phi=20,theta=45)

## End(Not run)
```

`cockroachAlData` *Spike Trains of several Cockroach Antennal Lobe Neurons Recorded from Three Animals*

Description

Four (`CAL1S` and `CAL1V`), three (`CAL2S` and `CAL2C`) and four (`e070528spont` and `e070528citronellal`) Cockroach (*Periplaneta americana*) antennal lobe neurons (putative projection neurons) were recorded simultaneously and extracellularly during spontaneous activity and odor (vanilin and citronellal) responses from three different animals. The data sets contain the sorted spike trains of the neurons.

Usage

```
data(CAL1S)
data(CAL1V)
data(CAL2S)
data(CAL2C)
data(e070528spont)
data(e070528citronellal)
```

Format

`CAL1S` is a named list with 4 components (`"neuron 1"`, `"neuron 2"`, `"neuron 3"`, `"neuron 4"`). Each component contains the spike train (ie, action potentials occurrence times) of one neuron recorded during 30 s of spontaneous activity. *Times are expressed in seconds.*

`CAL1V` is a named list with 4 components (`"neuron 1"`, `"neuron 2"`, `"neuron 3"`, `"neuron 4"`). Each component is a named list with 20 components: `"stim. 1"`, ..., `"stim. 20"`. Each sub-list contains the spike train of one neuron during 1 stimulation (odor puff) with vanilin. Each acquisition was 10 s long. The command to the odor delivery valve was on between sec 4.49 and sec 4.99.

`CAL2S` is a named list with 3 components (`"neuron 1"`, `"neuron 2"`, `"neuron 3"`). Each component contains the spike train (ie, action potentials occurrence times) of one neuron recorded during 1 mn of spontaneous activity. *Times are expressed in seconds.*

`CAL2C` is a named list with 3 components (`"neuron 1"`, `"neuron 2"`, `"neuron 3"`). Each component is a named list with 20 components: `"stim. 1"`, ..., `"stim. 20"`. Each sub-list contains the spike train of one neuron during 1 stimulation (odor puff) with vanilin. Each acquisition was 14 s long. The command to the odor delivery valve was on between sec 5.87 and sec 6.37.

`e070528spont` is a named list of with 4 components (`"neuron 1"`, `"neuron 2"`, `"neuron 3"`, `"neuron 4"`). Each component is a `spikeTrain` object (ie, action potentials occurrence times) of one neuron recorded during 60 s of spontaneous activity. *Times are expressed in seconds.*

`e070528citronellal` is a named list with 4 components (`"neuron 1"`, `"neuron 2"`, `"neuron 3"`, `"neuron 4"`). Each component is a `repeatedTrain` object with 15 `spikeTrain` objects: `"stim. 1"`, ..., `"stim. 15"`. Each `spikeTrain` contains the spike train of one neuron during 1 stimulation (odor puff) with citronellal. Each acquisition was 13 s long. The command to the odor delivery valve was on between sec 6.14 and sec 6.64.

Details

The data were recorded from neighboring sites on a *NeuroNexus* (http://neuronexustech.com/) silicon probe. Sorting was done with SpikeOMatic with superposition resolution which can AND DOES lead to artifcats on cross-correlograms.

Source

Recording and spike sorting performed by Antoine Chaffiol ⟨antoine.chaffiol@univ-paris5.fr⟩ at the Cerebral Physiology Lab, CNRS UMR 8118: http://www.biomedicale.univ-paris5.fr/physcerv/physiologie_cerebrale.htm.

References

http://www.biomedicale.univ-paris5.fr/physcerv/C_Pouzat/Doc/ChaffiolEtAl_FENS2006.pdf

Examples

```
## load CAL1S data
data(CAL1S)
## convert the data into spikeTrain objects
CAL1S <- lapply(CAL1S,as.spikeTrain)
## look at the train of the 1sd neuron
CAL1S[["neuron 1"]]
## fit the 6 different renewal models to the 1st neuron spike train
compModels(CAL1S[["neuron 1"]])
## look at the ISI distribution with the fitted invgauss dist for
## this 1st neuron
isiHistFit(CAL1S[["neuron 1"]],model="invgauss")

## load CAL1V data
data(CAL1V)
## convert them to repeatedTrain objects
CAL1V <- lapply(CAL1V, as.repeatedTrain)
## look at the raster of the 1st neuron
CAL1V[["neuron 1"]]

## load e070528spont data
data(e070528spont)
## look at the spike train of the 1st neuron
e070528spont[["neuron 1"]]

## load e070528citronellal data
data(e070528citronellal)
## look at the raster of the 1st neuron
plot(e070528citronellal[["neuron 1"]],stim=c(6.14,6.64))
```

---

coef.durationFit *Utility Functions for durationFit Objects*

Description

`coef.durationFit` and `logLik.durationFit` extract components of a `durationFit` object, while `is.durationFit` tests if its argument is such an object.

Usage

```
## S3 method for class 'durationFit':
coef(object,...)
## S3 method for class 'durationFit':
logLik(object,...)
is.durationFit(obj)
```

Arguments

`object`	a `durationFit` object.
`obj`	an object to be tested against a `durationFit` object.
`...`	see `coef` and `logLik`.

Details

Everything is trivial here.

Value

`coef.durationFit` returns the coefficients or the estimates or the fitted parameters of the object: a 2 elements named vector.

`logLik.durationFit` returns the loglikelihood value.

`is.durationFit` returns `TRUE` if its argument is a `durationFit` object and `FALSE` otherwise.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`compModels`, `invgaussMLE`, `lnormMLE`, `llogisMLE`, `rexpMLE`, `gammaMLE`, `weibullMLE`

Examples

```
## Not run:
## load CAL1S data
data(CAL1S)
## convert the data into spikeTrain objects
CAL1S <- lapply(CAL1S,as.spikeTrain)
## look at the train of the 1sd neuron
CAL1S[["neuron 1"]]
## fit a invgauss model to the 1st neuron spike train
n1SDFig <- invgaussMLE(CAL1S[["neuron 1"]])
is.durationFit(n1SDFig)
coef(n1SDFig)
logLik(n1SDFig)
## End(Not run)
```

compModels — *Compare Duration Models on a Specific Data Set*

Description

Fit duration models with the maximum likelihood method to a given duration data set. The data can be censored. The models should be among the following list: inverse Gaussian, log normal, log logistic, gamma, Weibull, refractory exponential. The Akaike information criterion (AIC) is used to produce a numerical output. Diagnostic QQ or survival plots can also be generated.

Usage

```
compModels(yi, ni = numeric(length(yi)) + 1,
           si = numeric(length(yi)) + 1,
           models = c("invgauss","lnorm","gamma","weibull","llogis","rexp"),
           type = c("qq","s"), log = TRUE, plot = TRUE)
```

Arguments

`yi`	vector of (possibly binned) observations or a `spikeTrain` object.
`ni`	vector of counts for each value of `yi`; default: `numeric(length(yi))+1`.
`si`	vector of counts of *uncensored* observations for each value of `yi`; default: `numeric(length(yi))+1`.
`models`	a character vector whose elements are selected among: `"invgauss"`, `"lnorm"`, `"gamma"`, `"weibull"`, `"llogis"`, `"rexp"`.
`type`	should a QQ plot (`"qq"`) or a survival plot (`"s"`) be generated?
`log`	should a log scale be used?
`plot`	should a plot be generated?

Details

Fits are performed by maximizing the likelihood.

Value

A vector whose component are nammed according to the model used and ordered along increasing AIC values.

if argument `plot` is set to `TRUE` (the default), a plot is generated as a side effect.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Lindsey, J.K. (2004) *The Statistical Analysis of Stochastic Processes in Time*. CUP.

See Also

`qqDuration`, `invgaussMLE`, `lnormMLE`, `llogisMLE`, `rexpMLE`, `gammaMLE`, `weibullMLE`

Examples

```
## Not run:
## load spontaneous data of 4 putative projection neurons
## simultaneously recorded from the cockroach (Periplaneta
## americana) antennal lobe
data(CAL1S)
## convert data into spikeTrain objects
CAL1S <- lapply(CAL1S,as.spikeTrain)
## look at the individual trains
## first the "raw" data
CAL1S[["neuron 1"]]
## next some summary information
summary(CAL1S[["neuron 1"]])
## next the renewal tests
renewalTestPlot(CAL1S[["neuron 1"]])
## It does not look too bad so let fit simple models
compModels(CAL1S[["neuron 1"]])

## Simulate a sample with 100 events from an inverse Gaussian
set.seed(1102006,"Mersenne-Twister")
mu.true <- 0.075
sigma2.true <- 3
sampleSize <- 100
sampIG <- rinvgauss(sampleSize,mu=mu.true,sigma2=sigma2.true)

## Compare models and display QQ plot
compModels(sampIG,type="qq")

## Compare models and display survival plot
compModels(sampIG,type="s")

## Generate a censored sample using an exponential distribution
sampEXP <- rexp(sampleSize,1/(2*mu.true))
sampIGtime <- pmin(sampIG,sampEXP)
sampIGstatus <- as.numeric(sampIG <= sampEXP)
## Compare models and display QQ plot
## WARNING with censored data like here the QQ plot is misleading
compModels(yi=sampIGtime,si=sampIGstatus,type="qq")
## Compare models and display survival plot
compModels(yi=sampIGtime,si=sampIGstatus,type="s")
## End(Not run)
```

---

df4counts — *Generates a Data Frame from a repeatedTrain Object After Time Binning*

Description

Generates a `data.frame` object out of a `repeatedTrain` object after time binning in order to study trials stationarity with a `glm` fit.

Usage

```
df4counts(repeatedTrain, breaks = length(repeatedTrain))
```

Arguments

`repeatedTrain`	a `repeatedTrain` object or a list which can be coerced to such an object.
`breaks`	a numeric. A single number is interpreted has the number of bins; a vector is interpreted as the position of the "breaks" between bins.

Details

The bins are placed between the `floor` of the smallest spike time and the `ceiling` of the largest one when `breaks` is a scalar. After time binning the number of spikes of each trial falling in each bin is counted (in the same way as the `counts` component of a `psth` list is obtained). This matrix of count is then formatted as a data frame.

Value

A `data.frame` with the following variables:

`Count`	a count (number of spikes in a given bin at a given trial).
`Bin`	the bin index (a `factor`.
`Trial`	the trial index (a `factor`.
`Rate`	the count divided by the length of the corresponding bin.
`Time`	the time of the midpoints of the bins.

Note

When a `glm` of the poisson family is used for subsequent analysis the important implicit hypothesis of an inhomogenous Poisson train is of course made.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`as.repeatedTrain`, `psth`

Examples

```
## Load the Vanillin responses of the first
## cockroach data set
data(CAL1V)
## convert them into repeatedTrain objects
## The stimulus command is on between 4.49 s and 4.99s
CAL1V <- lapply(CAL1V,as.repeatedTrain)
## Generate raster plot for neuron 1
raster(CAL1V[["neuron 1"]],c(4.49,4.99))
## make a smooth PSTH of these data
psth(CAL1V[["neuron 1"]],stimTimeCourse=c(4.49,4.99),breaks=c(bw=0.5,step=0.05),colCI=2,x
## add a grid to the plot
grid()
## The response starts after 4.5 s and is mostly over after 6 s: create
## breaks accordingly
myBreaks <- c(0,2.25,4.5,seq(4.75,6.25,0.25),seq(6.5,11,0.5))
```

```
## get a count data frame
CAL1Vn1DF <- df4counts(CAL1V[["neuron 1"]],myBreaks)
## use a box plot to look at the result
boxplot(Rate ~ Time, data=CAL1Vn1DF)
## watch out here the time scale is distorted because of our
## choice of unequal bins
## Fit a glm of the Poisson family taking both Bin and Trial effects
CAL1Vn1DFglm <- glm(Count ~ Bin + Trial,family=poisson,data=CAL1Vn1DF)
## use an anova to see that both the Bin effect and the trial effect are
## highly significant
anova(CAL1Vn1DFglm, test="Chisq")
```

`diff.spikeTrain` *diff method for spikeTrain objects*

Description

`diff` method for `spikeTrain` objects.

Usage

```
## S3 method for class 'spikeTrain':
diff(x, ...)
```

Arguments

`x`	a `spikeTrain` object.
`...`	see `diff`

Value

a `numeric`

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`diff`, `as.spikeTrain`, `is.spikeTrain`

Examples

```
data(CAL1S)
## convert data into spikeTrain objects
CAL1S <- lapply(CAL1S,as.spikeTrain)
## look at the individual trains
## first the "raw" data
CAL1S[["neuron 1"]]
## get the isi of neuron 1
n1.isi <- diff(CAL1S[["neuron 1"]])
```

`dinvgauss` *The Inverse Gaussian Distribution*

Description

Density, distribution function, quantile function, and random generation for the inverse Gaussian.

Usage

```
dinvgauss(x, mu = 1, sigma2 = 1, boundary = NULL,
          log = FALSE)
pinvgauss(q, mu = 1, sigma2 = 1,
          boundary = NULL, lower.tail = TRUE,
          log.p = FALSE)
qinvgauss(p, mu = 1, sigma2 = 1,
          boundary = NULL)
rinvgauss(n = 1, mu = 1, sigma2 = 1, boundary = NULL)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n) > 1`, the length is taken to be the number required.
`mu`	mean value of the distribution in the default parameterization, `mean value / boundary` otherwise. Can also be viewed as the inverse of the drift of the latent Brownian motion.
`sigma2`	variance of the latent Brownian motion. When this parameterization is used (the default) the distance between the "starting" point and the boundary ("absorbing barrier") is set to 1.
`boundary`	distance between the starting point and the "absorbing barrier" of the latent Brownian motion. When this parameterization is used the Brownian motion variance is set to 1.
`lower.tail`	logical; if `TRUE` (default), probabilities are `P[X <= x]`, otherwise, `P[X > x]`.
`log, log.p`	logical; if `TRUE`, probabilities p are given as log(p).

Details

With the default, `"sigma2"`, parameterization (`mu = m, sigma2 = s^2`) the inverse Gaussian distribution has density:

$$f(x) = \frac{1}{\sqrt{2\,\pi\,\sigma^2\,x^3}}\;\exp(-\frac{1}{2}\frac{(x-\mu)^2}{x\,\sigma^2\,\mu^2})$$

with $\sigma^2 > 0$. The theoretical mean is: μ and the theoretical variance is: $\mu^3\sigma^2$. With the default, `"boundary"`, parameterization (`mu = m, boundary = b`)the inverse Gaussian distribution has density:

$$f(x) = \frac{b}{\sqrt{2\,\pi\,x^3}}\;\exp(-\frac{1}{2}\frac{(x-b\,\mu)^2}{x\,\mu^2})$$

with $\sigma^2 > 0$. The theoretical mean is: $\mu\, b$ and the theoretical variance is: $\mu^3\sigma^2$. The latent Brownian motion is described in Lindsey (2004) pp 209-213, Whitemore and Seshadri (1987), Aalen and Gjessing (2001) and Gerstein and Mandelbrot (1964).

The expression for the distribution function is given in Eq. 4 of Whitemore and Seshadri (1987).

Initial guesses for the inversion of the distribution function used in `qinvgauss` are obtained with the transformation of Whitemore and Yalovsky (1978).

Random variates are obtained with the method of Michael et al (1976) which is also described by Devroye (1986, p 148) and Gentle (2003, p 193).

Value

`dinvgauss` gives the density, `pinvgauss` gives the distribution function, `qinvgauss` gives the quantile function and `rinvgauss` generates random deviates.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Gerstein, George L. and Mandelbrot, Benoit (1964) Random Walk Models for the Spike Activity of a Single Neuron. *Biophys J.* **4**: 41–68. http://www.pubmedcentral.nih.gov/articlerender.fcgi?tool=pubmed\&pubmedid=14104072.

Whitemore, G. A. and Yalovsky, M. (1978) A normalizing logarithmic transformation for inverse Gaussian random variables. *Technometrics* **20**: 207–208.

Whitmore, G. A. and Seshadri, V. (1987) A Heuristic Derivation of the Inverse Gaussian Distribution. *The American Statistician* **41**: 280–281.

Aalen, Odd O. and Gjessing, Hakon K. (2001) Understanding the Shape of the Hazard Rate: A Process Point of View. *Statistical Science* **16**: 1–14.

Lindsey, J.K. (2004) *Introduction to Applied Statistics: A Modelling Approach*. OUP.

Michael, J. R., Schucany, W. R. and Haas, R. W. (1976) Generating random variates using transformations with multiple roots. *The American Statistician* **30**: 88–90.

Devroye, L. (1986) *Non-Uniform Random Variate Generation*. Springer-Verlag. http://cg.scs.carleton.ca/~luc/rnbookindex.html.

Gentle, J. E. (2003) *Random Number Generation and Monte Carlo Methods*. Springer.

See Also

`invgaussMLE`, `Lognormal`, `hinvgauss`

Examples

```
## Not run:
## Start with the inverse Gauss
## Define standard mu and sigma
mu.true <- 0.075 ## a mean ISI of 75 ms
sigma2.true <- 3
## Define a sequence of points on the time axis
X <- seq(0.001,0.3,0.001)
## look at the density
plot(X,dinvgauss(X,mu.true,sigma2.true),type="l",xlab="ISI (s)",ylab="Density")
```

```
## Generate a sample of 100 ISI from this distribution
sampleSize <- 100
sampIG <- rinvgauss(sampleSize,mu=mu.true,sigma2=sigma2.true)
## check out the empirical survival function (obtained with the Kaplan-Meyer
## estimator) against the true one
library(survival)
sampIG.KMfit <- survfit(Surv(sampIG,1+numeric(length(sampIG))) ~1)
plot(sampIG.KMfit,log=TRUE)
lines(X,pinvgauss(X,mu.true,sigma2.true,lower.tail=FALSE),col=2)

## Get a ML fit
sampIGmleIG <- invgaussMLE(sampIG)
## compare true and estimated parameters
rbind(est = sampIGmleIG$estimate,se = sampIGmleIG$se,true = c(mu.true,sigma2.true))
## plot contours of the log relative likelihood function
Mu <- seq(sampIGmleIG$estimate[1]-3*sampIGmleIG$se[1],
          sampIGmleIG$estimate[1]+3*sampIGmleIG$se[1],
          sampIGmleIG$se[1]/10)
Sigma2 <- seq(sampIGmleIG$estimate[2]-7*sampIGmleIG$se[2],
              sampIGmleIG$estimate[2]+7*sampIGmleIG$se[2],
              sampIGmleIG$se[2]/10)
sampIGmleIGcontour <- sapply(Mu, function(mu) sapply(Sigma2, function(s2) sampIGmleIG$r(m
contour(Mu,Sigma2,t(sampIGmleIGcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels=c("log(0.5)",
          "log(0.1)",
          "-1/2*P(Chi2=0.95)",
          "-1/2*P(Chi2=0.99)"),
        xlab=expression(mu),ylab=expression(sigma^2))
points(mu.true,sigma2.true,pch=16,col=2)
## We can see that the contours are more parabola like on a log scale
contour(log(Mu),log(Sigma2),t(sampIGmleIGcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels=c("log(0.5)",
          "log(0.1)",
          "-1/2*P(Chi2=0.95)",
          "-1/2*P(Chi2=0.99)"),
        xlab=expression(log(mu)),ylab=expression(log(sigma^2)))
points(log(mu.true),log(sigma2.true),pch=16,col=2)
## make a deviance test for the true parameters
pchisq(-2*sampIGmleIG$r(mu.true,sigma2.true),df=2)
## check fit with a QQ plot
qqDuration(sampIGmleIG,log="xy")

## Generate a censored sample using an exponential distribution
sampEXP <- rexp(sampleSize,1/(2*mu.true))
sampIGtime <- pmin(sampIG,sampEXP)
sampIGstatus <- as.numeric(sampIG <= sampEXP)
## fit the censored sample
sampIG2mleIG <- invgaussMLE(sampIGtime,,sampIGstatus)
## look at the results
rbind(est = sampIG2mleIG$estimate,
      se = sampIG2mleIG$se,
      true = c(mu.true,sigma2.true))
pchisq(-2*sampIG2mleIG$r(mu.true,sigma2.true),df=2)
## repeat the survival function estimation
```

```
sampIG2.KMfit <- survfit(Surv(sampIGtime,sampIGstatus) ~1)
plot(sampIG2.KMfit,log=TRUE)
lines(X,pinvgauss(X,sampIG2mleIG$estimate[1],sampIG2mleIG$estimate[2],lower.tail=FALSE),c
## End(Not run)
```

`dllogis` *The Log Logistic Distribution*

Description

Density, distribution function, quantile function, and random generation for the log logistic.

Usage

```
dllogis(x, location = 0, scale = 1, log = FALSE)
pllogis(q, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
qllogis(p, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
rllogis(n, location = 0, scale = 1)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n) > 1`, the length is taken to be the number required.
`location, scale`	location and scale parameters (non-negative numeric).
`lower.tail`	logical; if `TRUE` (default), probabilities are `P[X <= x]`, otherwise, `P[X > x]`.
`log, log.p`	logical; if `TRUE`, probabilities p are given as log(p).

Details

If `location` or `scale` are omitted, they assume the default values of 0 and 1 respectively.

The log-Logistic distribution with `location = m` and `scale = s` has distribution function

$$\mathrm{F}(x) = \frac{1}{1+\exp(-\frac{\log(x)-m}{s})}$$

and density

$$\mathrm{f}(x) = \frac{1}{s\,x}\frac{\exp(-\frac{\log(x)-m}{s})}{(1+\exp(-\frac{\log(x)-m}{s}))^2}$$

Value

`dllogis` gives the density, `pllogis` gives the distribution function, `qllogis` gives the quantile function and `rllogis` generates random deviates.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Lindsey, J.K. (2004) *Introduction to Applied Statistics: A Modelling Approach*. OUP.

Lindsey, J.K. (2004) *The Statistical Analysis of Stochastic Processes in Time*. CUP.

See Also

`llogisMLE`, `Lognormal`, `hllogis`

Examples

```
## Not run:
tSeq <- seq(0.001,0.6,0.001)
location.true <- -2.7
scale.true <- 0.025
Yd <- dllogis(tSeq, location.true, scale.true)
Yh <- hllogis(tSeq, location.true, scale.true)
max.Yd <- max(Yd)
max.Yh <- max(Yh)
Yd <- Yd / max.Yd
Yh <- Yh / max.Yh
oldpar <- par(mar=c(5,4,4,4))
plot(tSeq, Yd, type="n", axes=FALSE, ann=FALSE,
     xlim=c(0,0.6), ylim=c(0,1))
axis(2,at=seq(0,1,0.2),labels=round(seq(0,1,0.2)*max.Yd,digits=2))
mtext("Density (1/s)", side=2, line=3)
axis(1,at=pretty(c(0,0.6)))
mtext("Time (s)", side=1, line=3)
axis(4, at=seq(0,1,0.2), labels=round(seq(0,1,0.2)*max.Yh,digits=2))
mtext("Hazard (1/s)", side=4, line=3, col=2)
mtext("Log Logistic Density and Hazard Functions", side=3, line=2,cex=1.5)
lines(tSeq,Yd)
lines(tSeq,Yh,col=2)
par(oldpar)
## End(Not run)
```

`drexp` *The Refractory Exponential Distribution*

Description

Density, distribution function, quantile function, and random generation for the refractory exponential.

Usage

```
drexp(x, rate = 10, rp = 0.005, log = FALSE)
prexp(q, rate = 10, rp = 0.005, lower.tail = TRUE, log.p = FALSE)
qrexp(p, rate = 10, rp = 0.005, lower.tail = TRUE, log.p = FALSE)
rrexp(n, rate = 10, rp = 0.005)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n) > 1`, the length is taken to be the number required.
`lower.tail`	logical; if `TRUE` (default), probabilities are `P[X <= x]`, otherwise, `P[X > x]`.
`log, log.p`	logical; if `TRUE`, probabilities p are given as log(p).
`rate`	rate parameter (non-negative numeric).
`rp`	refractory period parameter (non-negative numeric).

Details

The refractory exponential distribution with `rate`, r, and `refractory period`, rp, has density:

f(x) = r exp(- r (x-rp))

for `x >= rp`.

Value

`drexp` gives the density, `prexp` gives the distribution function, `qrexp` gives the quantile function and `rrexp` generates random deviates.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Johnson, D. H. and Swami, A. (1983) The transmission of signals by auditory-nerve fiber discharge patterns. *J. Acoust. Soc. Am.* **74**: 493–501.

See Also

`rexpMLE`

Examples

```
## Not run:
tSeq <- seq(0.001,0.6,0.001)
rate.true <- 20
rp.true <- 0.01
Yd <- drexp(tSeq, rate.true, rp.true)
Yh <- hrexp(tSeq, rate.true, rp.true)
max.Yd <- max(Yd)
max.Yh <- max(Yh)
Yd <- Yd / max.Yd
Yh <- Yh / max.Yh
oldpar <- par(mar=c(5,4,4,4))
plot(tSeq, Yd, type="n", axes=FALSE, ann=FALSE,
     xlim=c(0,0.6), ylim=c(0,1))
axis(2,at=seq(0,1,0.2),labels=round(seq(0,1,0.2)*max.Yd,digits=2))
mtext("Density (1/s)", side=2, line=3)
```

```
axis(1,at=pretty(c(0,0.6)))
mtext("Time (s)", side=1, line=3)
axis(4, at=seq(0,1,0.2), labels=round(seq(0,1,0.2)*max.Yh,digits=2))
mtext("Hazard (1/s)", side=4, line=3, col=2)
mtext("Refractory Exponential Density and Hazard Functions", side=3, line=2,cex=1.5)
lines(tSeq,Yd)
lines(tSeq,Yh,col=2)
par(oldpar)
## End(Not run)
```

---

frt — *Computes Forward Recurrence Times from Two transformedTrain Objects*

Description

Computes the (transformed) time differences between spikes of a `refTrain` and the (next) ones of a `testTrain`. Both `refTrain` and `testTrain` should be `transformedTrain` objects.

Usage

```
frt(refTrain, testTrain)
refTrain %frt% testTrain
```

Arguments

`refTrain`	a `transformedTrain` object.
`testTrain`	a `transformedTrain` object.

Details

When two spike trains have been time transformed using *the same* procedure, which does make one of the trains (the `testTrain`) the realization a homogeneous Poisson process with rate 1, the elapsed time between the spikes of the other train (`refTrain`) and the ones of `testTrain` should be exponentially distributed with rate 1. These elapsed times are returned by `frt`.

Value

An object of class `frt` containing the elapsed times.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`transformedTrain`, `plot.frt`, `summary.frt`, `mkGLMdf`

Examples

```
## Not run:
## Let us consider neuron 1 of the CAL2S data set
data(CAL2S)
CAL2S <- lapply(CAL2S,as.spikeTrain)
CAL2S[["neuron 1"]]
renewalTestPlot(CAL2S[["neuron 1"]])
summary(CAL2S[["neuron 1"]])
## Make a data frame with a 4 ms time resolution
cal2Sdf <- mkGLMdf(CAL2S,0.004,0,60)
## keep the part relative to neuron 1, 2 and 3 separately
n1.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="1",]
n2.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="2",]
n3.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="3",]
## remove unnecessary data
rm(cal2Sdf)
## Extract the elapsed time since the second to last and
## third to last for neuron 1. Normalise the result.
n1.cal2sDF[c("rlN.1","rsN.1","rtN.1")] <- brt4df(n1.cal2sDF,"lN.1",2,c("rlN.1","rsN.1","r
## load mgcv library
library(mgcv)
## fit a model with a tensorial product involving the last
## three spikes and using a cubic spline basis for the last two
## To gain time use a fixed df regression spline
n1S.fitA <- gam(event ~ te(rlN.1,rsN.1,bs="cr",fx=TRUE) + rtN.1,data=n1.cal2sDF,family=bi
## transform time
N1.Lambda <- transformedTrain(n1S.fitA)
## check out the resulting spike train using the fact
## that transformedTrain objects inherit from spikeTrain
## objects
N1.Lambda
## Use more formal checks
summary(N1.Lambda)
plot(N1.Lambda,which=c(1,2,4,5),ask=FALSE)
## Transform spike trains of neuron 2 and 3
N2.Lambda <- transformedTrain(n1S.fitA,n2.cal2sDF$event)
N3.Lambda <- transformedTrain(n1S.fitA,n3.cal2sDF$event)
## Check interactions
summary(N2.Lambda %frt% N1.Lambda)
summary(N3.Lambda %frt% N1.Lambda)
plot(N2.Lambda %frt% N1.Lambda,ask=FALSE)
plot(N3.Lambda %frt% N1.Lambda,ask=FALSE)
## End(Not run)
```

`gammaMLE` *Maximum Likelihood Parameter Estimation of a Gamma Model with Possibly Censored Data*

Description

Estimate Gamma model parameters by the maximum likelihood method using possibly censored data. Two different parameterizations of the Gamma distribution can be used.

Usage

```
gammaMLE(yi, ni = numeric(length(yi)) + 1,
         si = numeric(length(yi)) + 1, scale = TRUE)
```

Arguments

`yi`	vector of (possibly binned) observations or a `spikeTrain` object.
`ni`	vector of counts for each value of `yi`; default: `numeric(length(yi))+1`.
`si`	vector of counts of *uncensored* observations for each value of `yi`; default: `numeric(length(yi))+1`.
`scale`	logical should the scale (`TRUE`) or the rate parameterization (`FALSE`) be used?

Details

There is no closed form expression for the MLE of a Gamma distribution. The numerical method implemented here uses the profile likelihood described by Monahan (2001) pp 210-216.

In order to ensure good behavior of the numerical optimization routines, optimization is performed on the log of the parameters (`shape` and `scale` or `rate`).

Standard errors are obtained from the inverse of the observed information matrix at the MLE. They are transformed to go from the log scale used by the optimization routine to the parameterization requested.

Value

A list of class `durationFit` with the following components:

`estimate`	the estimated parameters, a named vector.
`se`	the standard errors, a named vector.
`logLik`	the log likelihood at maximum.
`r`	a function returning the log of the relative likelihood function.
`mll`	a function returning the opposite of the log likelihood function using the log of the parameters.
`call`	the matched call.

Note

The returned standard errors (component `se`) are valid in the asymptotic limit. You should plot contours using function `r` in the returned list and check that the contours are reasonably close to ellipses.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Monahan, J. F. (2001) *Numerical Methods of Statistics*. CUP.

Lindsey, J.K. (2004) *Introduction to Applied Statistics: A Modelling Approach*. OUP.

See Also

GammaDist, invgaussMLE, lnormMLE

Examples

```
## Not run:
## Simulate sample of size 100 from a gamma distribution
set.seed(1102006,"Mersenne-Twister")
sampleSize <- 100
shape.true <- 6
scale.true <- 0.012
sampGA <- rgamma(sampleSize,shape=shape.true,scale=scale.true)
sampGAmleGA <- gammaMLE(sampGA)
rbind(est = sampGAmleGA$estimate,se = sampGAmleGA$se,true = c(shape.true,scale.true))

## Estimate the log relative likelihood on a grid to plot contours
Shape <- seq(sampGAmleGA$estimate[1]-4*sampGAmleGA$se[1],
               sampGAmleGA$estimate[1]+4*sampGAmleGA$se[1],
               sampGAmleGA$se[1]/10)
Scale <- seq(sampGAmleGA$estimate[2]-4*sampGAmleGA$se[2],
             sampGAmleGA$estimate[2]+4*sampGAmleGA$se[2],
             sampGAmleGA$se[2]/10)
sampGAmleGAcontour <- sapply(Shape, function(sh) sapply(Scale, function(sc) sampGAmleGA$r
## plot contours using a linear scale for the parameters
## draw four contours corresponding to the following likelihood ratios:
##  0.5, 0.1, Chi2 with 2 df and p values of 0.95 and 0.99
X11(width=12,height=6)
layout(matrix(1:2,ncol=2))
contour(Shape,Scale,t(sampGAmleGAcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels=c("log(0.5)",
          "log(0.1)",
          "-1/2*P(Chi2=0.95)",
          "-1/2*P(Chi2=0.99)"),
        xlab="shape",ylab="scale",
        main="Log Relative Likelihood Contours"
        )
points(sampGAmleGA$estimate[1],sampGAmleGA$estimate[2],pch=3)
points(shape.true,scale.true,pch=16,col=2)
## The contours are not really symmetrical about the MLE we can try to
## replot them using a log scale for the parameters to see if that improves
## the situation
contour(log(Shape),log(Scale),t(sampGAmleGAcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels="",
        xlab="log(shape)",ylab="log(scale)",
        main="Log Relative Likelihood Contours",
        sub="log scale for the parameters")
points(log(sampGAmleGA$estimate[1]),log(sampGAmleGA$estimate[2]),pch=3)
points(log(shape.true),log(scale.true),pch=16,col=2)

## make a parametric boostrap to check the distribution of the deviance
nbReplicate <- 10000
sampleSize <- 100
system.time(
            devianceGA100 <- replicate(nbReplicate,{
```

```
                                sampGA <- rgamma(sampleSize,shape=shape.true,scale=scale.tru
                                sampGAmleGA <- gammaMLE(sampGA)
                                -2*sampGAmleGA$r(shape.true,scale.true)
                              }
                                          )
                 )[3]

## Get 95 and 99% confidence intervals for the QQ plot
ci <- sapply(1:nbReplicate,
                 function(idx) qchisq(qbeta(c(0.005,0.025,0.975,0.995),
                                              idx,
                                              nbReplicate-idx+1),
                                        df=2)
              )
## make QQ plot
X <- qchisq(ppoints(nbReplicate),df=2)
Y <- sort(devianceGA100)
X11()
plot(X,Y,type="n",
     xlab=expression(paste(chi[2]^2," quantiles")),
     ylab="MC quantiles",
     main="Deviance with true parameters after ML fit of gamma data",
     sub=paste("sample size:", sampleSize,"MC replicates:", nbReplicate)
     )
abline(a=0,b=1)
lines(X,ci[1,],lty=2)
lines(X,ci[2,],lty=2)
lines(X,ci[3,],lty=2)
lines(X,ci[4,],lty=2)
lines(X,Y,col=2)
## End(Not run)
```

---

gamObj — *Generic Function and Methods for Extracting a gamObject*

Description

Some functions of STAR, like spsth and slockedTrain perform gam fits internally and keep as a list component or within the environment of a returned function the result of this fit. Method gamObj extracts this gam object.

Usage

```
gamObj(object, ...)
## S3 method for class 'spsth':
gamObj(object, ...)
## S3 method for class 'slockedTrain':
gamObj(object, ...)
```

Arguments

object	an object containing a gamObject. Currently the result of a call to spsth or to slockedTrain.
...	not used for now

Value

A gamObject

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

gam, gamObject, spsth, slockedTrain

Examples

```
##
```

---

hgamma *Hazard Functions for Some Common Duration Distributions*

Description

Hazard functions for the gamma, weibull, lognormal, inverse Gaussian, log logistic and refractory exponential distributions

Usage

```
hgamma(x, shape, rate = 1, scale = 1/rate, log = FALSE)
hweibull(x, shape, scale = 1, log = FALSE)
hlnorm(x, meanlog = 0, sdlog = 1, log = FALSE)
hinvgauss(x, mu = 1, sigma2 = 1, boundary = NULL,
          log = FALSE)
hllogis(x, location = 0, scale = 1, log = FALSE)
hrexp(x, rate = 10, rp = 0.005, log = FALSE)
```

Arguments

`x`	vector of quantiles.
`shape, scale, rate, sdlog`	strictly positive parameters. See corresponding distributions for detail.
`mu, sigma2, boundary`	parameters associated with the inverse Gaussian distribution.
`meanlog`	parameter associated with the log normal distribution.
`location, rp`	parameters of the log logistic and refratory exponential.
`log`	should the log hazard be returned? `FALSE` by default.

Details

These functions are simply obtained by deviding the density by the survival fucntion.

Value

A vector of hazard rates.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Lindsey, J.K. (2004) *Introduction to Applied Statistics: A Modelling Approach*. OUP.

Lindsey, J.K. (2004) *The Statistical Analysis of Stochastic Processes in Time*. CUP.

See Also

`dinvgauss`, `dllogis`, `drexp`

Examples

```
## Not run:
## use a few plots to compare densities and hazard functions

## lognormal
tSeq <- seq(0.001,0.6,0.001)
meanlog.true <- -2.4
sdlog.true <- 0.4
Yd <- dlnorm(tSeq,meanlog.true,sdlog.true)
Yh <- hlnorm(tSeq,meanlog.true,sdlog.true)
max.Yd <- max(Yd)
max.Yh <- max(Yh)
Yd <- Yd / max.Yd
Yh <- Yh / max.Yh
oldpar <- par(mar=c(5,4,4,4))
plot(tSeq, Yd, type="n", axes=FALSE, ann=FALSE,
     xlim=c(0,0.6), ylim=c(0,1))
axis(2,at=seq(0,1,0.2),labels=round(seq(0,1,0.2)*max.Yd,digits=2))
mtext("Density (1/s)", side=2, line=3)
axis(1,at=pretty(c(0,0.6)))
mtext("Time (s)", side=1, line=3)
axis(4, at=seq(0,1,0.2), labels=round(seq(0,1,0.2)*max.Yh,digits=2))
mtext("Hazard (1/s)", side=4, line=3, col=2)
mtext("Lognormal Density and Hazard Functions", side=3, line=2,cex=1.5)
lines(tSeq,Yd)
lines(tSeq,Yh,col=2)
par(oldpar)

## inverse Gaussian
tSeq <- seq(0.001,0.6,0.001)
mu.true <- 0.075
sigma2.true <- 3
Yd <- dinvgauss(tSeq,mu.true,sigma2.true)
Yh <- hinvgauss(tSeq,mu.true,sigma2.true)
max.Yd <- max(Yd)
max.Yh <- max(Yh)
Yd <- Yd / max.Yd
Yh <- Yh / max.Yh
oldpar <- par(mar=c(5,4,4,4))
plot(tSeq, Yd, type="n", axes=FALSE, ann=FALSE,
     xlim=c(0,0.6), ylim=c(0,1))
axis(2,at=seq(0,1,0.2),labels=round(seq(0,1,0.2)*max.Yd,digits=2))
mtext("Density (1/s)", side=2, line=3)
```

```
axis(1,at=pretty(c(0,0.6)))
mtext("Time (s)", side=1, line=3)
axis(4, at=seq(0,1,0.2), labels=round(seq(0,1,0.2)*max.Yh,digits=2))
mtext("Hazard (1/s)", side=4, line=3, col=2)
mtext("Inverse Gaussian Density and Hazard Functions", side=3, line=2,cex=1.5)
lines(tSeq,Yd)
lines(tSeq,Yh,col=2)
par(oldpar)

## gamma
tSeq <- seq(0.001,0.6,0.001)
shape.true <- 6
scale.true <- 0.012
Yd <- dgamma(tSeq, shape=shape.true, scale=scale.true)
Yh <- hgamma(tSeq, shape=shape.true, scale=scale.true)
max.Yd <- max(Yd)
max.Yh <- max(Yh)
Yd <- Yd / max.Yd
Yh <- Yh / max.Yh
oldpar <- par(mar=c(5,4,4,4))
plot(tSeq, Yd, type="n", axes=FALSE, ann=FALSE,
     xlim=c(0,0.6), ylim=c(0,1))
axis(2,at=seq(0,1,0.2),labels=round(seq(0,1,0.2)*max.Yd,digits=2))
mtext("Density (1/s)", side=2, line=3)
axis(1,at=pretty(c(0,0.6)))
mtext("Time (s)", side=1, line=3)
axis(4, at=seq(0,1,0.2), labels=round(seq(0,1,0.2)*max.Yh,digits=2))
mtext("Hazard (1/s)", side=4, line=3, col=2)
mtext("Gamma Density and Hazard Functions", side=3, line=2,cex=1.5)
lines(tSeq,Yd)
lines(tSeq,Yh,col=2)
par(oldpar)

## Weibull
tSeq <- seq(0.001,0.6,0.001)
shape.true <- 2.5
scale.true <- 0.085
Yd <- dweibull(tSeq, shape=shape.true, scale=scale.true)
Yh <- hweibull(tSeq, shape=shape.true, scale=scale.true)
max.Yd <- max(Yd)
max.Yh <- max(Yh)
Yd <- Yd / max.Yd
Yh <- Yh / max.Yh
oldpar <- par(mar=c(5,4,4,4))
plot(tSeq, Yd, type="n", axes=FALSE, ann=FALSE,
     xlim=c(0,0.6), ylim=c(0,1))
axis(2,at=seq(0,1,0.2),labels=round(seq(0,1,0.2)*max.Yd,digits=2))
mtext("Density (1/s)", side=2, line=3)
axis(1,at=pretty(c(0,0.6)))
mtext("Time (s)", side=1, line=3)
axis(4, at=seq(0,1,0.2), labels=round(seq(0,1,0.2)*max.Yh,digits=2))
mtext("Hazard (1/s)", side=4, line=3, col=2)
mtext("Weibull Density and Hazard Functions", side=3, line=2,cex=1.5)
lines(tSeq,Yd)
lines(tSeq,Yh,col=2)
par(oldpar)
```

```
## refractory exponential
tSeq <- seq(0.001,0.6,0.001)
rate.true <- 20
rp.true <- 0.01
Yd <- drexp(tSeq, rate.true, rp.true)
Yh <- hrexp(tSeq, rate.true, rp.true)
max.Yd <- max(Yd)
max.Yh <- max(Yh)
Yd <- Yd / max.Yd
Yh <- Yh / max.Yh
oldpar <- par(mar=c(5,4,4,4))
plot(tSeq, Yd, type="n", axes=FALSE, ann=FALSE,
     xlim=c(0,0.6), ylim=c(0,1))
axis(2,at=seq(0,1,0.2),labels=round(seq(0,1,0.2)*max.Yd,digits=2))
mtext("Density (1/s)", side=2, line=3)
axis(1,at=pretty(c(0,0.6)))
mtext("Time (s)", side=1, line=3)
axis(4, at=seq(0,1,0.2), labels=round(seq(0,1,0.2)*max.Yh,digits=2))
mtext("Hazard (1/s)", side=4, line=3, col=2)
mtext("Refractory Exponential Density and Hazard Functions", side=3, line=2,cex=1.5)
lines(tSeq,Yd)
lines(tSeq,Yh,col=2)
par(oldpar)

## log logistic
tSeq <- seq(0.001,0.6,0.001)
location.true <- -2.7
scale.true <- 0.025
Yd <- dllogis(tSeq, location.true, scale.true)
Yh <- hllogis(tSeq, location.true, scale.true)
max.Yd <- max(Yd)
max.Yh <- max(Yh)
Yd <- Yd / max.Yd
Yh <- Yh / max.Yh
oldpar <- par(mar=c(5,4,4,4))
plot(tSeq, Yd, type="n", axes=FALSE, ann=FALSE,
     xlim=c(0,0.6), ylim=c(0,1))
axis(2,at=seq(0,1,0.2),labels=round(seq(0,1,0.2)*max.Yd,digits=2))
mtext("Density (1/s)", side=2, line=3)
axis(1,at=pretty(c(0,0.6)))
mtext("Time (s)", side=1, line=3)
axis(4, at=seq(0,1,0.2), labels=round(seq(0,1,0.2)*max.Yh,digits=2))
mtext("Hazard (1/s)", side=4, line=3, col=2)
mtext("Log Logistic Density and Hazard Functions", side=3, line=2,cex=1.5)
lines(tSeq,Yd)
lines(tSeq,Yh,col=2)
par(oldpar)
## End(Not run)
```

`hist.lockedTrain` *Auto- and Cross-Intensity Function Estimate for Spike Trains*

Description

`hist.lockedTrain` constructs and `plot.hist.lockedTrain` plots estimates of what Cox and Lewis (1966) call the auto- or cross-intensity functions. The auto-intensity function is also called the renewal density by Cox and Lewis and by Perkel et al (1967) while it is called the intensity of the Palm distribution by Ogata (1988). The (estimate of) the cross-intensity function is called cross-correlation function by Perkel et al (1967b) and cross-correlation histogram by Brillinger et al (1976).

Usage

```
## S3 method for class 'lockedTrain':
hist(x, bw, breaks = NULL, plot = TRUE, ...)
## S3 method for class 'hist.lockedTrain':
plot(x, style = c("Ogata", "Brillinger"),
                 CI, unit = "s", xlab, ylab, xlim, ylim,
                 type, pch, ...)
```

Arguments

`x`	a `lockedTrain` object returned by the `lockedTrain` function.
`bw`	a non-negative numeric, the bin width.
`breaks`	a vector giving the breakpoints between cells. If `NULL` (default) breaks are built using argument `bw` and component `laglim` of `obj`.
`plot`	a logical. If `TRUE` a plot is generated as a side effect and nothing is returned, if `FALSE` a list of class `hist.lockedTime` is returned.
`style`	a character. The style of the plot, `"Ogata"` or `"Brillinger"`.
`CI`	a numeric vector with at most two elements. The coverage probability of the confidence intervals.
`unit`	a character. The unit in which the spike times are expressed.
`xlab`	a character. The x label. Default supplied.
`ylab`	a character. The y label. Default supplied.
`xlim`	a numeric. See `plot`. Default supplied.
`ylim`	a numeric. See `plot`. Default supplied.
`type`	see `lines`. Default supplied.
`pch`	see `plot`. Default supplied.
`...`	see `plot`.

Details

The intensity of the Palm distribution (Ogata, 1988, p 13) is estimated by:

$$\mathrm{m}(s) = \frac{\mathrm{Prob}(\text{event in } (t+s, t+s+\Delta s) \mid \text{event at } t)}{\Delta s}$$

It is called *renewal density* by Perkel et al (1967) and defined by their Eq. 10, p 404. Under the null hypothesis of a stationary Poisson process it is a constant whose value is the mean discharge rate.

The cross-intensity function of two spike trains A and B is estimated by (Perkel et al, 1967b, p424, Eq. 4 and 5):

$$\mathrm{m}_{AB}(s) = \frac{\mathrm{Prob}(\text{B event in } (t+s, t+s+\Delta s) \mid \text{A event at } t)}{\Delta s}$$

The `style` argument of `plot.hist.lockedTrain` generates a plot looking like Fig. 6, p 18 of Ogata (1988) if set to `"Ogata"`. Using `style` `"Brillinger"` plots the square root of the estimate.

Value

When argument `plot` in `hist.lockedTrain` is set to `FALSE` a list of class `hist.lockedTrain` with the following components is returned:

`density`	the density estimate. Equivalent of the component `density` returned by `hist`.
`breaks`	a numeric vector with the breaks in between which spikes were counted. Similar to the component of the same name returned by `hist`.
`mids`	a numeric vector with the mid points of `breaks`. . Similar to the component of the same name returned by `hist`.
`bw`	the bin width used.
`nRef`	the total number of reference spikes used.
`refFreq`	the mean frequency of the reference neuron.
`testFreq`	the mean frequency of the test neuron.
`obsTime`	the total observation time used (in s).
`CCH`	a logical which is `TRUE` if a cross-intensity was estimated and `FALSE` in the case of an auto-intensity.
`call`	the matched call.

Note

The confidence intervals are obtained from a Poisson distribution with parameter: `refFreq` * `testFreq` * `bw` * `obsTime`. Once the quantiles of the Poisson distribution have been obtained they are divided by: `refFreq` * `bw` * `obsTime`

These intervals are valid under the stationary Poisson null hypothesis for the auto-intensity estimates. They are valid under the *stationary independent* null hypothesis for the cross-intensity. *There is NO NEED to assume that the test train is Poisson or renewal.* See Perkel et al (1967b) and McFadden (1962) for a justification/proof of that. The square root transform of Brillinger (1976) and Brillinger et al (1976) is (in my opinion) a perfect example of shooting at a sparrow with a bazooka. An oversized method to get at an intuitively obvious result. There is moreover no need to stabilize the variance if your testing against a Poisson with a constant rate since then the variance of the null hypothesis is stable to start with. These (square root) transforms are useful for least square fits with a Poisson noise but NOT in the present context.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Ogata, Yosihiko (1988) Statistical Models for Earthquake Occurrences and Residual Analysis for Point Processes. *Journal of the American Statistical Association* **83**: 9-27.

D. R. Cox and P. A. W. Lewis (1966) *The Statistical Analysis of Series of Events*. John Wiley and Sons.

J. A. McFadden (1962) On the Lengths of Intervals in a Stationary Point Process. *Journal of the Royal Statistical Society. Series B*, **24**: 364-382

Perkel D. H., Gerstein, G. L. and Moore G. P. (1967) Neural Spike Trains and Stochastic Point Processes. I. The Single Spike Train. *Biophys. J.*, **7**: 391-418. `http://www.pubmedcentral.nih.gov/articlerender.fcgi?tool=pubmed\&pubmedid=4292791`

Perkel D. H., Gerstein, G. L. and Moore G. P. (1967b) Neural Spike Trains and Stochastic Point Processes. I. Simultaneous Spike Trains. *Biophys. J.*, **7**: 419-440. http://www.pubmedcentral.nih.gov/articlerender.fcgi?tool=pubmed\&pubmedid=4292792

David R. Brillinger, Hugh L. Bryant and Jose P. Segundo (1976) Identification of synaptic interactions. *Biol Cybern*, **22**: 213-228.

David R. Brillinger (1976) Estimation of the Second-Order Intensities of a Bivariate Stationary Point Process. *Journal of the Royal Statistical Society. Series B (Methodological)*, **38**, 60-66.

See Also

`varianceTime`, `renewalTestPlot`, `lockedTrain`

Examples

```
## Reproduce Fig. 6 of Ogata 1988
data(ShallowShocks)
shalShocks <- lockedTrain(as.spikeTrain(ShallowShocks$Date),,c(0,500))
shalShocksH <- hist(shalShocks,5,plot=FALSE)
plot(shalShocksH,"Ogata",c(0.95,0.99),xlab="TIME LAG (DAYS)",ylab="NUMBER OF EVENTS PER D
## Reproduce Fig. 7 of Ogata 1988
myBinNb <- 101
myMidPoints <- seq(from = 1, to = 6, length.out=myBinNb)
myMidPoints <- 10^myMidPoints/200
myBreaks <- c(0,myMidPoints[-myBinNb] + diff(myMidPoints) / 2)
shalShocksH2 <- hist(shalShocks,breaks=myBreaks,plot=FALSE)
yy <- abs(shalShocksH2$density - shalShocksH2$refFreq)
plot(shalShocksH2$mids[shalShocksH2$density>0],
     yy[shalShocksH2$density>0],
     pch = 1,
     xlim = c(0.001,10000),
     log = "xy",
     xlab = "TIME LAG (DAYS)",
     ylab = "NUMBER OF EVENTS PER DAY"
     )
```

---

invgaussMLE *Maximum Likelihood Parameter Estimation of an Inverse Gaussian Model with Possibly Censored Data*

Description

Estimate inverse Gaussian model parameters by the maximum likelihood method using possibly censored data. Two different parameterizations of the inverse Gaussian distribution can be used.

Usage

```
invgaussMLE(yi, ni = numeric(length(yi)) + 1,
             si = numeric(length(yi)) + 1,
             parameterization = "sigma2")
```

Arguments

`yi`	vector of (possibly binned) observations or a `spikeTrain` object.
`ni`	vector of counts for each value of `yi`; default: `numeric(length(yi))+1`.
`si`	vector of counts of *uncensored* observations for each value of `yi`; default: `numeric(length(yi))+1`.
`parameterization`	parameterization used, `"sigma2"` (default) of `"boundary"`.

Details

The 2 different parameterizations of the inverse Gaussian distribution are discussed in the manual of `dinvgauss`.

In the absence of censored data the ML estimates are available in closed form (Lindsey, 2004, p 212) together with the Hessian matrix at the MLE. In presence of censored data an initial guess for the parameters is obtained using the uncensored data before maximizing the likelihood function to the full data set using `optim` with the `BFGS` method. ML estimation is always performed with the `"sigma2"` parameterization. Parameters and variance-covariance matrix are transformed at the end if the `"boundary"` parameterization is requested.

In order to ensure good behavior of the numerical optimization routines, optimization is performed on the log of the parameters (`mu` and `sigma2`).

Standard errors are obtained from the inverse of the observed information matrix at the MLE. They are transformed to go from the log scale used by the optimization routine, when the latter is used (ie, for censored data) to the parameterization requested.

Value

A list of class `durationFit` with the following components:

`estimate`	the estimated parameters, a named vector.
`se`	the standard errors, a named vector.
`logLik`	the log likelihood at maximum.
`r`	a function returning the log of the relative likelihood function.
`mll`	a function returning the opposite of the log likelihood function using the log of the parameters.
`call`	the matched call.

Note

The returned standard errors (component `se`) are valid in the asymptotic limit. You should plot contours using function `r` in the returned list and check that the contours are reasonably close to ellipses.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Lindsey, J.K. (2004) *Introduction to Applied Statistics: A Modelling Approach*. OUP.

See Also

dinvgauss,lnormMLE,gammaMLE,weibullMLE,llogisMLE,rexpMLE.

Examples

```
## Simulate sample of size 100 from an inverse Gaussian
## distribution
set.seed(1102006,"Mersenne-Twister")
sampleSize <- 100
mu.true <- 0.075
sigma2.true <- 3
sampleSize <- 100
sampIG <- rinvgauss(sampleSize,mu=mu.true,sigma2=sigma2.true)
## Make a maximum likelihood fit
sampIGmleIG <- invgaussMLE(sampIG)
## Compare estimates with actual values
rbind(est = coef(sampIGmleIG),se = sampIGmleIG$se,true = c(mu.true,sigma2.true))
## In the absence of censoring the MLE of the inverse Gaussian is available in a
## closed form together with its variance (ie, the observed information matrix)
## we can check that we did not screw up at that stage by comparing the observed
## information matrix obtained numerically with the analytical one. To do that we
## use the MINUS log likelihood function returned by invgaussMLE to get a numerical
## estimate
detailedFit <- optim(par=as.vector(log(sampIGmleIG$estimate)),
                     fn=sampIGmleIG$mll,
                     method="BFGS",
                     hessian=TRUE)
## We should not forget that the "mll" function uses the log of the parameters while
## the "se" component of sampIGmleIG list is expressed on the linear scale we must theref
## transform one into the other as follows (Kalbfleisch, 1985, p71):
## if x = exp(u) and y = exp(v) and if we have the information matrix in term of
## u and v (that's the hessian component of list detailedFit above), we create matrix:
##      du/dx du/dy
## Q =
##      dv/dx dv/dy
## and we get I in term of x and y by the following matrix product:
## I(x,y) <- t(Q) %*% I(u,v) %*% Q
## In the present case:
##  du/dx = 1/exp(u), du/dy = 0, dv/dx = 0, dv/dy = 1/exp(v)
## Therefore:
Q <- diag(1/exp(detailedFit$par))
numericalI <- t(Q) %*% detailedFit$hessian %*% Q
seComp <- rbind(sampIGmleIG$se, sqrt(diag(solve(numericalI))))
colnames(seComp) <- c("mu","sigma2")
rownames(seComp) <- c("analytical", "numerical")
seComp
## We can check the relative differences between the 2
apply(seComp,2,function(x) abs(diff(x))/x[1])

## Not run:
## Estimate the log relative likelihood on a grid to plot contours
Mu <- seq(coef(sampIGmleIG)[1]-4*sampIGmleIG$se[1],
          coef(sampIGmleIG)[1]+4*sampIGmleIG$se[1],
          sampIGmleIG$se[1]/10)
Sigma2 <- seq(coef(sampIGmleIG)[2]-4*sampIGmleIG$se[2],
              coef(sampIGmleIG)[2]+4*sampIGmleIG$se[2],
                 sampIGmleIG$se[2]/10)
sampIGmleIGcontour <- sapply(Mu, function(mu) sapply(Sigma2, function(s2) sampIGmleIG$r(m
## plot contours using a linear scale for the parameters
## draw four contours corresponding to the following likelihood ratios:
##  0.5, 0.1, Chi2 with 2 df and p values of 0.95 and 0.99
X11(width=12,height=6)
layout(matrix(1:2,ncol=2))
contour(Mu,Sigma2,t(sampIGmleIGcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels=c("log(0.5)",
          "log(0.1)",
          "-1/2*P(Chi2=0.95)",
          "-1/2*P(Chi2=0.99)"),
        xlab=expression(mu),ylab=expression(sigma^2),
        main="Log Relative Likelihood Contours"
        )
points(coef(sampIGmleIG)[1],coef(sampIGmleIG)[2],pch=3)
points(mu.true,sigma2.true,pch=16,col=2)
## The contours are not really symmetrical about the MLE we can try to
## replot them using a log scale for the parameters to see if that improves
## the situation
contour(log(Mu),log(Sigma2),t(sampIGmleIGcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels="",
        xlab=expression(log(mu)),ylab=expression(log(sigma^2)),
        main="Log Relative Likelihood Contours",
        sub="log scale for the parameters")
points(log(coef(sampIGmleIG)[1]),log(coef(sampIGmleIG)[2]),pch=3)
points(log(mu.true),log(sigma2.true),pch=16,col=2)

## Even with the log scale the contours are not ellipsoidal, so let us compute profiles
## For that we are going to use the returned MINUS log likelihood function
logMuProfFct <- function(logMu,...) {
  myOpt <- optimise(function(x) sampIGmleIG$mll(c(logMu,x))+logLik(sampIGmleIG),...)
  as.vector(unlist(myOpt[c("objective","minimum")]))
}
logMuProfCI <- function(logMu,
                        CI,
                        a=logS2Seq[1],
                        b=logS2Seq[length(logS2Seq)]) logMuProfFct(logMu,c(a,b))[1] - qch

logS2ProfFct <- function(logS2,...) {
  myOpt <- optimise(function(x) sampIGmleIG$mll(c(x,logS2))+logLik(sampIGmleIG),...)
  as.vector(unlist(myOpt[c("objective","minimum")]))
}
logS2ProfCI <- function(logS2, CI,
                        a=logMuSeq[1],
                        b=logMuSeq[length(logMuSeq)]) logS2ProfFct(logS2,c(a,b))[1] - qch

## We compute profiles (on the log scale) eploxing +/- 3 times
## the se about the MLE
logMuSE <- sqrt(diag(solve(detailedFit$hessian)))[1]
logMuSeq <- seq(log(coef(sampIGmleIG)[1])-3*logMuSE,
                log(coef(sampIGmleIG)[1])+3*logMuSE,
                logMuSE/10)
logS2SE <- sqrt(diag(solve(detailedFit$hessian)))[2]
logS2Seq <- seq(log(coef(sampIGmleIG)[2])-3*logS2SE,
                log(coef(sampIGmleIG)[2])+3*logS2SE,
                logS2SE/10)
logMuProf <- sapply(logMuSeq,logMuProfFct,
                    lower=logS2Seq[1],
                    upper=logS2Seq[length(logS2Seq)])
## Get 95
logMuCI95 <- c(uniroot(logMuProfCI,
                       interval=c(logMuSeq[1],log(coef(sampIGmleIG)[1])),
                       CI=0.95)$root,
               uniroot(logMuProfCI,
                       interval=c(log(coef(sampIGmleIG)[1]),logMuSeq[length(logMuSeq)]),
                       CI=0.95)$root
               )
logMuCI99 <- c(uniroot(logMuProfCI,
                       interval=c(logMuSeq[1],log(coef(sampIGmleIG)[1])),
                       CI=0.99)$root,
               uniroot(logMuProfCI,
                       interval=c(log(coef(sampIGmleIG)[1]),logMuSeq[length(logMuSeq)]),
                       CI=0.99)$root
               )

logS2Prof <- sapply(logS2Seq,logS2ProfFct,
                    lower=logMuSeq[1],
                    upper=logMuSeq[length(logMuSeq)])
## Get 95
logS2CI95 <- c(uniroot(logS2ProfCI,
                       interval=c(logS2Seq[1],log(coef(sampIGmleIG)[2])),
                       CI=0.95)$root,
               uniroot(logS2ProfCI,
                       interval=c(log(coef(sampIGmleIG)[2]),logS2Seq[length(logS2Seq)]),
                       CI=0.95)$root
               )
logS2CI99 <- c(uniroot(logS2ProfCI,
                       interval=c(logS2Seq[1],log(coef(sampIGmleIG)[2])),
                       CI=0.99)$root,
               uniroot(logS2ProfCI,
                       interval=c(log(coef(sampIGmleIG)[2]),logS2Seq[length(logS2Seq)]),
                       CI=0.99)$root
               )

## Add profiles to the previous plot
lines(logMuSeq,logMuProf[2,],col=2,lty=2)
lines(logS2Prof[2,],logS2Seq,col=2,lty=2)

## We can now check the deviations of the (profiled) deviances
## from the asymptotic parabolic curves
X11()
layout(matrix(1:4,nrow=2))
oldpar <- par(mar=c(4,4,2,1))
logMuSeqOffset <- logMuSeq-log(coef(sampIGmleIG)[1])
logMuVar <- diag(solve(detailedFit$hessian))[1]
plot(logMuSeq,2*logMuProf[1,],type="l",xlab=expression(log(mu)),ylab="Deviance")
lines(logMuSeq,logMuSeqOffset^2/logMuVar,col=2)
points(log(coef(sampIGmleIG)[1]),0,pch=3)
abline(h=0)
abline(h=qchisq(0.95,1),lty=2)
abline(h=qchisq(0.99,1),lty=2)
```

```
lines(rep(logMuCI95[1],2),c(0,qchisq(0.95,1)),lty=2)
lines(rep(logMuCI95[2],2),c(0,qchisq(0.95,1)),lty=2)
lines(rep(logMuCI99[1],2),c(0,qchisq(0.99,1)),lty=2)
lines(rep(logMuCI99[2],2),c(0,qchisq(0.99,1)),lty=2)
## We can also "linearize" this last graph
plot(logMuSeq,
     sqrt(2*logMuProf[1,])*sign(logMuSeqOffset),
     type="l",
     xlab=expression(log(mu)),
     ylab=expression(paste("signed ",sqrt(Deviance)))
     )
lines(logMuSeq,
      sqrt(logMuSeqOffset^2/logMuVar)*sign(logMuSeqOffset),
      col=2)
points(log(coef(sampIGmleIG)[1]),0,pch=3)

logS2SeqOffset <- logS2Seq-log(coef(sampIGmleIG)[2])
logS2Var <- diag(solve(detailedFit$hessian))[2]
plot(logS2Seq,2*logS2Prof[1,],type="l",xlab=expression(log(sigma^2)),ylab="Deviance")
lines(logS2Seq,logS2SeqOffset^2/logS2Var,col=2)
points(log(coef(sampIGmleIG)[2]),0,pch=3)
abline(h=0)
abline(h=qchisq(0.95,1),lty=2)
abline(h=qchisq(0.99,1),lty=2)
lines(rep(logS2CI95[1],2),c(0,qchisq(0.95,1)),lty=2)
lines(rep(logS2CI95[2],2),c(0,qchisq(0.95,1)),lty=2)
lines(rep(logS2CI99[1],2),c(0,qchisq(0.99,1)),lty=2)
lines(rep(logS2CI99[2],2),c(0,qchisq(0.99,1)),lty=2)
## We can also "linearize" this last graph
plot(logS2Seq,
     sqrt(2*logS2Prof[1,])*sign(logS2SeqOffset),
     type="l",
     xlab=expression(log(sigma^2)),
     ylab=expression(paste("signed ",sqrt(Deviance)))
     )
lines(logS2Seq,
      sqrt(logS2SeqOffset^2/logS2Var)*sign(logS2SeqOffset),
      col=2)
points(log(coef(sampIGmleIG)[2]),0,pch=3)
par(oldpar)

## make a parametric boostrap to check the distribution of the deviance
nbReplicate <- 1000 #10000
sampleSize <- 100
system.time(
deviance IG100 <- lapply(1:nbReplicate,
                        function(idx) {
                          if ((idx
                          sampIG <- rinvgauss(sampleSize,mu=mu.true,sigma2=sigma2.true)
                          sampIGmleIG <- invgaussMLE(sampIG)
                          Deviance <- -2*sampIGmleIG$r(mu.true,sigma2.true)
                          logPara <- log(coef(sampIGmleIG))
                          logParaSE <- sampIGmleIG$se/coef(sampIGmleIG)
                          intervalMu <- function(n) c(-n,n)*logParaSE[1]+logPara[1]
                          intervalS2 <- function(n) c(-n,n)*logParaSE[2]+logPara[2]
                          logMuProfFct <- function(logMu,...) {
                            optimise(function(x)
           sampIGmleIG$mll(c(logMu,x))+logLik(sampIGmleIG),...)
}
logMuProfCI <- function(logMu,
                        CI,
                        a=intervalS2(4)[1],
                        b=intervalS2(4)[2])
  logMuProfFct(logMu,c(a,b)) - qchisq(CI,1)/2

logS2ProfFct <- function(logS2,...) {
  optimise(function(x)
           sampIGmleIG$mll(c(x,logS2))+logLik(sampIGmleIG),...)
}
logS2ProfCI <- function(logS2, CI,
                        a=intervalMu(4)[1],
                        b=intervalMu(4)[2])
  logS2ProfFct(logS2,c(a,b)) - qchisq(CI,1)/2

factor <- 4
while((logMuProfCI(intervalMu(factor)[2],0.99) *
       logMuProfCI(logPara[1],0.99) >= 0) ||
      (logMuProfCI(intervalMu(factor)[1],0.99) *
       logMuProfCI(logPara[1],0.99) >= 0)
      ) factor <- factor+1
##browser()
logMuCI95 <- c(uniroot(logMuProfCI,
                       interval=c(intervalMu(factor)[1],logPara
                       CI=0.95)$root,
               uniroot(logMuProfCI,
                       interval=c(logPara[1],intervalMu(factor)
                       CI=0.95)$root
               )
logMuCI99 <- c(uniroot(logMuProfCI,
                       interval=c(intervalMu(factor)[1],logPara
                       CI=0.99)$root,
               uniroot(logMuProfCI,
                       interval=c(logPara[1],intervalMu(factor)
                       CI=0.99)$root
               )
factor <- 4
while((logS2ProfCI(intervalS2(factor)[2],0.99) *
       logS2ProfCI(logPara[2],0.99) >= 0) ||
      (logS2ProfCI(intervalS2(factor)[1],0.99) *
       logS2ProfCI(logPara[2],0.99) >= 0)
      ) factor <- factor+1
logS2CI95 <- c(uniroot(logS2ProfCI,
                       interval=c(intervalS2(factor)[1],logPara
                       CI=0.95)$root,
               uniroot(logS2ProfCI,
                          interval=c(logPara[2],intervalS2(fact
                       CI=0.95)$root
               )
logS2CI99 <- c(uniroot(logS2ProfCI,
                       interval=c(intervalS2(factor)[1],logPara
                       CI=0.99)$root,
               uniroot(logS2ProfCI,
                       interval=c(logPara[2],intervalS2(factor)
                       CI=0.99)$root
```

```
                                          )
                            list(deviance=Deviance,
                                 logMuCI95=logMuCI95,
                                 logMuNorm95=qnorm(c(0.025,0.975),logPara[1],logParaSE[1]),
                                 logMuCI99=logMuCI99,
                                 logMuNorm99=qnorm(c(0.005,0.995),logPara[1],logParaSE[1]),
                                 logS2CI95=logS2CI95,
                                 logS2Norm95=qnorm(c(0.025,0.975),logPara[2],logParaSE[2]),
                                 logS2CI99=logS2CI99,
                                 logS2Norm99=qnorm(c(0.005,0.995),logPara[2],logParaSE[2]))
                          }
                          )
            )[3]
## Find out how many times the true parameters was within the computed CIs
nLogMuCI95 <- sum(sapply(devianceIG100,
                         function(l) l$logMuCI95[1] <= log(mu.true)  &&
                         log(mu.true)<= l$logMuCI95[2]
                         )
                  )
nLogMuNorm95 <- sum(sapply(devianceIG100,
                           function(l) l$logMuNorm95[1] <= log(mu.true)  &&
                           log(mu.true)<= l$logMuNorm95[2]
                           )
                    )
nLogMuCI99 <- sum(sapply(devianceIG100,
                         function(l) l$logMuCI99[1] <= log(mu.true)  &&
                         log(mu.true)<= l$logMuCI99[2]
                         )
                  )
nLogMuNorm99 <- sum(sapply(devianceIG100,
                           function(l) l$logMuNorm99[1] <= log(mu.true)  &&
                           log(mu.true)<= l$logMuNorm99[2]
                           )
                    )
## Check if these counts are compatible with the nominal CIs
c(prof95Mu=nLogMuCI95,norm95Mu=nLogMuNorm95)
qbinom(c(0.005,0.995),nbReplicate,0.95)
c(prof95Mu=nLogMuCI99,norm95Mu=nLogMuNorm99)
qbinom(c(0.005,0.995),nbReplicate,0.99)

nLogS2CI95 <- sum(sapply(devianceIG100,
                         function(l) l$logS2CI95[1] <= log(sigma2.true)  &&
                         log(sigma2.true)<= l$logS2CI95[2]
                         )
                  )
nLogS2Norm95 <- sum(sapply(devianceIG100,
                           function(l) l$logS2Norm95[1] <= log(sigma2.true)  &&
                           log(sigma2.true)<= l$logS2Norm95[2]
                           )
                    )
nLogS2CI99 <- sum(sapply(devianceIG100,
                         function(l) l$logS2CI99[1] <= log(sigma2.true)  &&
                         log(sigma2.true)<= l$logS2CI99[2]
                         )
                  )
nLogS2Norm99 <- sum(sapply(devianceIG100,
                           function(l) l$logS2Norm99[1] <= log(sigma2.true)  &&
                              log(sigma2.true)<= l$logS2Norm99[2]
                              )
                       )
## Check if these counts are compatible with the nominal CIs
c(prof95S2=nLogS2CI95,norm95S2=nLogS2Norm95)
qbinom(c(0.005,0.995),nbReplicate,0.95)
c(prof95S2=nLogS2CI99,norm95S2=nLogS2Norm99)
qbinom(c(0.005,0.995),nbReplicate,0.99)

## Get 95 and 99% confidence intervals for the QQ plot
ci <- sapply(1:nbReplicate,
                 function(idx) qchisq(qbeta(c(0.005,0.025,0.975,0.995),
                                             idx,
                                             nbReplicate-idx+1),
                                      df=2)
             )
## make QQ plot
X <- qchisq(ppoints(nbReplicate),df=2)
Y <- sort(sapply(devianceIG100,function(l) l$deviance))
X11()
plot(X,Y,type="n",
     xlab=expression(paste(chi[2]^2," quantiles")),
     ylab="MC quantiles",
     main="Deviance with true parameters after ML fit of IG data",
     sub=paste("sample size:", sampleSize,"MC replicates:", nbReplicate)
     )
abline(a=0,b=1)
lines(X,ci[1,],lty=2)
lines(X,ci[2,],lty=2)
lines(X,ci[3,],lty=2)
lines(X,ci[4,],lty=2)
lines(X,Y,col=2)
## End(Not run)
```

`isiHistFit` *ISI Histogram With Fitted Model and CI*

Description

Fits a duration model to isis from a spike train. Confidence intervals are also drawn.

Usage

```
isiHistFit(spikeTrain, model, nbins = 10, CI = 0.95, ...)
```

Arguments

`spikeTrain`	a `spikeTrain` object or a numeric vector that can be coerced to such an object.
`model`	a character vector whose elements are selected among: `"invgauss"`, `"lnorm"`, `"gamma"`, `"weibull"`, `"llogis"`, `"rexp"`.
`nbins`	the number of bins to use.
`CI`	the confidence coefficient.
`...`	additional arguments passed to `hist`, see `hist`.

Details

Assuming that the train is reasonably well described by a renewal process, a `model` distribution is fitted to the inter-spike intervals (isis) obtained from `spikeTrain`. The fitted distribution is then used to set the histogram breaks such that a uniform bin count would be expected if the fitted distribution was the true one. Confidence segments are also obtained from the binomial distribution. The histogram is build and the fitted density together with confidence intervals are drawn.

Value

Nothing returned, `isiHistFit` is used for its side effect, a plot is generated on the current graphic device.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`compModels`, `hist`

Examples

```
## Not run:
## load spontaneous data of 4 putative projection neurons
## simultaneously recorded from the cockroach (Periplaneta
## americana) antennal lobe
data(CAL1S)
## convert data into spikeTrain objects
CAL1S <- lapply(CAL1S,as.spikeTrain)
## look at the individual trains
## first the "raw" data
CAL1S[["neuron 1"]]
## next some summary information
summary(CAL1S[["neuron 1"]])
## next the renewal tests
renewalTestPlot(CAL1S[["neuron 1"]])
## It does not look too bad so let fit simple models
compModels(CAL1S[["neuron 1"]])
## the best one is the invgauss. Let's look at
## it in detail
isiHistFit(CAL1S[["neuron 1"]],"invgauss",xlim=c(0,0.5))
## End(Not run)
```

`jpsth` *Related Functions and Methods for Joint-PSTHs and Joint Scatter Diagrams*

Description

Some mainly graphical tools to probe interactions between 2 neurons recorded in the presence of a repeated stimulation.

Usage

```
jsd(xRT, yRT, acquisitionWindow, xlab, ylab,
    main, pch = ".", ...)
jpsth(xRT, yRT, xBreaks, yBreaks,
      acquisitionWindow, nbEvtPerBin = 50)
## S3 method for class 'jpsth':
contour(x, xlab, ylab, main, ...)
## S3 method for class 'jpsth':
image(x, xlab, ylab, main, ...)
## S3 method for class 'jpsth':
persp(x, xlab, ylab, main, ...)
jpsth2df(object)
```

Arguments

`xRT`	a `repeatedTrain` object whose spike times will appear on the abscissa of the plots.
`yRT`	a `repeatedTrain` object whose spike times will appear on the ordinate of the plots. It must have the same length as `xRT`.
`x, object`	`jpsth` objects.
`xBreaks, yBreaks`	A single number (the bin width) or a vector defining bins boundaries on the X and Y axis. If missing a default is provided.
`acquisitionWindow`	a 2 elements vector specifying the begining and the end of the acquisition. If missing values are obtained using the `floor` of the smallest spike time and the `ceiling` of the largest one.
`nbEvtPerBin`	If both `xBreaks` and `xBreaks` are missing a bin width, `bw`, is computed such that the expected value of the count per cell (2 dimensional bin) would be `nbEvtPerBin` assuming a stationary Poisson discharge for both neurons.
`xlab`	a character (default value supplied). See `plot`.
`ylab`	a character (default value supplied). See `plot`.
`main`	a character (default value supplied). See `plot`.
`pch`	the type of "points" displayed by `jsd`. See `plot`.
`...`	additional arguments passed to `plot` by `jsd` and to respective generic methods by `contour.jpsth`, `image.jpsth` and `persp.jpsth`.

Details

The joint scatter diagram was introduced by Gerstein and Perkel (1972). The joint peristimulus time histogram is a binned version of it (Aertsen et al, 1989). `jpsth2df` allows the reformating of a `jpsth` object in order to compute a smooth version of it with `gam`.

Value

`jsd` is used for its side effect, a plot is generated and nothing is returned.

`jpsth2df` returns a `data.frame` with the following variables: `Count`, the counts per cell; `X`, the position of the cell on the X axis; `Y`, the position of the cell on the Y axis; and attributes: `xBreaks`, `yBreaks`, `xTotal`, `yTotal`, `nbTrials`, `acquisitionWindow` corresponding to the components of its argument with the same name and `originalCall` corresponding to component `call`.

`jpsth` returns a list of class `jpsth` with the following components:

`counts`	a matrix storing the counts per cell.
`density`	a matrix storing the density in each cell.
`xMids`	a vector containing the X positions of the cells.
`yMids`	a vector containing the Y positions of the cells.
`xBreaks`	a vector containing the bin boundaries of the cells along the X axis.
`yBreaks`	a vector containing the bin boundaries of the cells along the X axis.
`xTotal`	the total number of spikes of the "X" neuron.
`yTotal`	the total number of spikes of the "Y" neuron.
`xFreq`	the mean freqency of the "X" neuron.
`yFreq`	the mean freqency of the "Y" neuron.
`nbTrials`	the number of trials of `xRT` (and `yRT`).
`acquisitionWindow`	the boundaries of the acquisition window.
`call`	the matched call.

Note

I use "joint scatter diagram" for what Gerstein and Perkel (1972) more properly call a "joint peristimulus time scatter diagram".

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Gerstein, G. L. and Perkel, D. H. (1972) Mutual temporal relationships among neuronal spike trains. Statistical techniques for display and analysis. *Biophys J* **12**: 453–473. `http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1484144`

Aertsen, A. M., Gerstein, G. L., Habib, M. K., Palm, G. (1989) Dynamics of neuronal firing correlation: modulation of "effective connectivity". *J Neurophysiol* **61**: 900–917. `http://jn.physiology.org/cgi/content/abstract/61/5/900`

See Also

`lockedTrain`, `plot.lockedTrain`, `hist.lockedTrain`, `slockedTrain`, `plot.slockedTrain`, `contour`, `image`, `persp`, `attr`, `attributes`

Examples

```
## load e070528citronellal data
data(e070528citronellal)
## plot a jsd with neuron 1 on X and neuron 2 on Y
jsd(e070528citronellal[[1]],e070528citronellal[[2]])
## now make the jpsth
j1.2 <- jpsth(e070528citronellal[[1]],e070528citronellal[[2]])
```

```
## make a contour plot
contour(j1.2)
## make an image plot
image(j1.2)
## make a persp plot
persp(j1.2)
## get a data frame
j1.2DF <- jpsth2df(j1.2)
## fit a gam model assuming no interaction
## Not run:
fitNoI <- gam(Count ~ s(X,k=100,bs="cr") + s(Y,k=100,bs="cr"),data=j1.2DF,family=poisson(
## End(Not run)
```

`llogisMLE` *Maximum Likelihood Parameter Estimation of a Log Logistic Model with Possibly Censored Data*

Description

Estimate log logistic model parameters by the maximum likelihood method using possibly censored data.

Usage

```
llogisMLE(yi, ni = numeric(length(yi)) + 1,
          si = numeric(length(yi)) + 1)
```

Arguments

`yi`	vector of (possibly binned) observations or a `spikeTrain` object.
`ni`	vector of counts for each value of `yi`; default: `numeric(length(yi))+1`.
`si`	vector of counts of *uncensored* observations for each value of `yi`; default: `numeric(length(yi))+1`.

Details

The MLE for the log logistic is not available in closed formed and is therefore obtained numerically obtained by calling `optim` with the `BFGS` method.

In order to ensure good behavior of the numerical optimization routines, optimization is performed on the log of parameter `scale`.

Standard errors are obtained from the inverse of the observed information matrix at the MLE. They are transformed to go from the log scale used by the optimization routine to the requested parameterization.

Value

A list of class `durationFit` with the following components:

`estimate`	the estimated parameters, a named vector.
`se`	the standard errors, a named vector.
`logLik`	the log likelihood at maximum.

`r`	a function returning the log of the relative likelihood function.
`mll`	a function returning the opposite of the log likelihood function using the log of parameter `sdlog`.
`call`	the matched call.

Note

The returned standard errors (component `se`) are valid in the asymptotic limit. You should plot contours using function `r` in the returned list and check that the contours are reasonably close to ellipses.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Lindsey, J.K. (2004) *Introduction to Applied Statistics: A Modelling Approach*. OUP.

Lindsey, J.K. (2004) *The Statistical Analysis of Stochastic Processes in Time*. CUP.

See Also

`dllogis`, `invgaussMLE`, `gammaMLE`, `weibullMLE`, `rexpMLE`, `lnormMLE`

Examples

```
## Not run:
## Simulate sample of size 100 from a log logisitic
## distribution
set.seed(1102006,"Mersenne-Twister")
sampleSize <- 100
location.true <- -2.7
scale.true <- 0.025
sampLL <- rllogis(sampleSize,location=location.true,scale=scale.true)
sampLLmleLL <- llogisMLE(sampLL)
rbind(est = sampLLmleLL$estimate,se = sampLLmleLL$se,true = c(location.true,scale.true))

## Estimate the log relative likelihood on a grid to plot contours
Loc <- seq(sampLLmleLL$estimate[1]-4*sampLLmleLL$se[1],
               sampLLmleLL$estimate[1]+4*sampLLmleLL$se[1],
               sampLLmleLL$se[1]/10)
Scale <- seq(sampLLmleLL$estimate[2]-4*sampLLmleLL$se[2],
             sampLLmleLL$estimate[2]+4*sampLLmleLL$se[2],
             sampLLmleLL$se[2]/10)
sampLLmleLLcontour <- sapply(Loc, function(m) sapply(Scale, function(s) sampLLmleLL$r(m,s
## plot contours using a linear scale for the parameters
## draw four contours corresponding to the following likelihood ratios:
##  0.5, 0.1, Chi2 with 2 df and p values of 0.95 and 0.99
X11(width=12,height=6)
layout(matrix(1:2,ncol=2))
contour(Loc,Scale,t(sampLLmleLLcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels=c("log(0.5)",
          "log(0.1)",
          "-1/2*P(Chi2=0.95)",
          "-1/2*P(Chi2=0.99)"),
        xlab="Location",ylab="Scale",
        main="Log Relative Likelihood Contours"
        )
points(sampLLmleLL$estimate[1],sampLLmleLL$estimate[2],pch=3)
points(location.true,scale.true,pch=16,col=2)
## The contours are not really symmetrical about the MLE we can try to
## replot them using a log scale for the parameters to see if that improves
## the situation
contour(Loc,log(Scale),t(sampLLmleLLcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels="",
        xlab="log(Location)",ylab="log(Scale)",
        main="Log Relative Likelihood Contours",
        sub="log scale for parameter: scale")
points(sampLLmleLL$estimate[1],log(sampLLmleLL$estimate[2]),pch=3)
points(location.true,log(scale.true),pch=16,col=2)

## make a parametric boostrap to check the distribution of the deviance
nbReplicate <- 10000
sampleSize <- 100
system.time(
            devianceLL100 <- replicate(nbReplicate,{
              sampLL <- rllogis(sampleSize,location=location.true,scale=scale.true)
              sampLLmleLL <- llogisMLE(sampLL)
              -2*sampLLmleLL$r(location.true,scale.true)
            }
                                       )
            )[3]

## Get 95 and 99
ci <- sapply(1:nbReplicate,
                 function(idx) qchisq(qbeta(c(0.005,0.025,0.975,0.995),
                                             idx,
                                             nbReplicate-idx+1),
                                       df=2)
             )
## make QQ plot
X <- qchisq(ppoints(nbReplicate),df=2)
Y <- sort(devianceLL100)
X11()
plot(X,Y,type="n",
     xlab=expression(paste(chi[2]^2," quantiles")),
     ylab="MC quantiles",
     main="Deviance with true parameters after ML fit of log logistic data",
     sub=paste("sample size:", sampleSize,"MC replicates:", nbReplicate)
     )
abline(a=0,b=1)
lines(X,ci[1,],lty=2)
lines(X,ci[2,],lty=2)
lines(X,ci[3,],lty=2)
lines(X,ci[4,],lty=2)
lines(X,Y,col=2)
## End(Not run)
```

lnormMLE — *Maximum Likelihood Parameter Estimation of a Log Normal Model with Possibly Censored Data*

Description

Estimate log normal model parameters by the maximum likelihood method using possibly censored data.

Usage

```
lnormMLE(yi, ni = numeric(length(yi)) + 1,
         si = numeric(length(yi)) + 1)
```

Arguments

`yi`	vector of (possibly binned) observations or a `spikeTrain` object.
`ni`	vector of counts for each value of `yi`; default: `numeric(length(yi))+1`.
`si`	vector of counts of *uncensored* observations for each value of `yi`; default: `numeric(length(yi))+1`.

Details

In the absence of censored data the ML estimates are available in closed form together with the Hessian matrix at the MLE. In presence of censored data an initial guess for the parameters is obtained using the uncensored data before maximizing the likelihood function to the full data set using `optim` with the `BFGS` method.

In order to ensure good behavior of the numerical optimization routines, optimization is performed on the log of parameter `sdlog`.

Standard errors are obtained from the inverse of the observed information matrix at the MLE. They are transformed to go from the log scale used by the optimization routine, when the latter is used (ie, for censored data) to the parameterization requested.

Value

A list of class `durationFit` with the following components:

`estimate`	the estimated parameters, a named vector.
`se`	the standard errors, a named vector.
`logLik`	the log likelihood at maximum.
`r`	a function returning the log of the relative likelihood function.
`mll`	a function returning the opposite of the log likelihood function using the log of parameter `sdlog`.
`call`	the matched call.

Note

The returned standard errors (component `se`) are valid in the asymptotic limit. You should plot contours using function `r` in the returned list and check that the contours are reasonably close to ellipses.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@univ-paris5.fr⟩

References

Lindsey, J.K. (2004) *Introduction to Applied Statistics: A Modelling Approach*. OUP.

See Also

`Lognormal`,`invgaussMLE`

Examples

```
## Simulate sample of size 100 from a log normal
## distribution
set.seed(1102006,"Mersenne-Twister")
sampleSize <- 100
meanlog.true <- -2.4
sdlog.true <- 0.4
sampLN <- rlnorm(sampleSize,meanlog.true,sdlog.true)
sampLNmleLN <- lnormMLE(sampLN)
rbind(est = sampLNmleLN$estimate,se = sampLNmleLN$se,true = c(meanlog.true,sdlog.true))
## In the absence of censoring the MLE of the log normal is available in a
## closed form together with its variance (ie, the observed information matrix)
## we can check that we did not screw up at that stage by comparing the observed
## information matrix obtained numerically with the analytical one. To do that we
## use the MINUS log likelihood function returned by lnormMLE to get a numerical
## estimate
detailedFit <- optim(fn=sampLNmleLN$mll,
                     par=as.vector(c(sampLNmleLN$estimate[1],log(sampLNmleLN$estimate[2])
                     method="BFGS",
                     hessian=TRUE)
## We should not forget that the "mll" function uses the log of the sdlog parameter while
## the "se" component of sampLNmleLN list is expressed on the linear scale we must theref
## transform one into the other as follows (Kalbfleisch, 1985, p71):
## if x = u and y = exp(v) and if we have the information matrix in term of
## u and v (that's the hessian component of list detailedFit above), we create matrix:
##       du/dx du/dy
## Q =
##       dv/dx dv/dy
## and we get I in term of x and y by the following matrix product:
## I(x,y) <- t(Q) %*% I(u,v) %*% Q
## In the present case:
##  du/dx = 1, du/dy = 0, dv/dx = 0, dv/dy = 1/exp(v)
## Therefore:
Q <- diag(c(1,1/exp(detailedFit$par[2])))
numericalI <- t(Q) %*% detailedFit$hessian %*% Q
seComp <- rbind(sampLNmleLN$se, sqrt(diag(solve(numericalI))))
colnames(seComp) <- c("meanlog","sdlog")
rownames(seComp) <- c("analytical", "numerical")
seComp
## We can check the relative differences between the 2
apply(seComp,2,function(x) abs(diff(x))/x[1])

## Not run:
## Estimate the log relative likelihood on a grid to plot contours
```

```
MeanLog <- seq(sampLNmleLN$estimate[1]-4*sampLNmleLN$se[1],
               sampLNmleLN$estimate[1]+4*sampLNmleLN$se[1],
               sampLNmleLN$se[1]/10)
SdLog <- seq(sampLNmleLN$estimate[2]-4*sampLNmleLN$se[2],
             sampLNmleLN$estimate[2]+4*sampLNmleLN$se[2],
             sampLNmleLN$se[2]/10)
sampLNmleLNcontour <- sapply(MeanLog, function(mu) sapply(SdLog, function(s) sampLNmleLN$
## plot contours using a linear scale for the parameters
## draw four contours corresponding to the following likelihood ratios:
##  0.5, 0.1, Chi2 with 2 df and p values of 0.95 and 0.99
X11(width=12,height=6)
layout(matrix(1:2,ncol=2))
contour(MeanLog,SdLog,t(sampLNmleLNcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels=c("log(0.5)",
          "log(0.1)",
          "-1/2*P(Chi2=0.95)",
          "-1/2*P(Chi2=0.99)"),
        xlab=expression(mu),ylab=expression(sigma),
        main="Log Relative Likelihood Contours"
        )
points(sampLNmleLN$estimate[1],sampLNmleLN$estimate[2],pch=3)
points(meanlog.true,sdlog.true,pch=16,col=2)
## The contours are not really symmetrical about the MLE we can try to
## replot them using a log scale for the parameters to see if that improves
## the situation
contour(MeanLog,log(SdLog),t(sampLNmleLNcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels="",
        xlab=expression(mu),ylab=expression(log(sigma)),
        main="Log Relative Likelihood Contours",
        sub=expression(paste("log scale for parameter: ",sigma)))
points(sampLNmleLN$estimate[1],log(sampLNmleLN$estimate[2]),pch=3)
points(meanlog.true,log(sdlog.true),pch=16,col=2)

## make a parametric boostrap to check the distribution of the deviance
nbReplicate <- 10000
sampleSize <- 100
system.time(
            devianceLN100 <- replicate(nbReplicate,{
                                 sampLN <- rlnorm(sampleSize,meanlog=meanlog.true,sdlog=sdlog
                                 sampLNmleLN <- lnormMLE(sampLN)
                                 -2*sampLNmleLN$r(meanlog.true,sdlog.true)
                               }
                                           )
            )[3]

## Get 95 and 99% confidence intervals for the QQ plot
ci <- sapply(1:nbReplicate,
                 function(idx) qchisq(qbeta(c(0.005,0.025,0.975,0.995),
                                             idx,
                                             nbReplicate-idx+1),
                                      df=2)
             )
## make QQ plot
X <- qchisq(ppoints(nbReplicate),df=2)
Y <- sort(devianceLN100)
```

```
X11()
plot(X,Y,type="n",
     xlab=expression(paste(chi[2]^2," quantiles")),
     ylab="MC quantiles",
     main="Deviance with true parameters after ML fit of logNorm data",
     sub=paste("sample size:", sampleSize,"MC replicates:", nbReplicate)
     )
abline(a=0,b=1)
lines(X,ci[1,],lty=2)
lines(X,ci[2,],lty=2)
lines(X,ci[3,],lty=2)
lines(X,ci[4,],lty=2)
lines(X,Y,col=2)
## End(Not run)
```

`lockedTrain` *Construct and Plot Time-Dependent Cross-correlation Diagram*

Description

`lockedTrain` constructs and `plot.lockedTrain` (and `print.lockedTrain`) plot what van Stokkum et al (1986) call a time-dependent cross-correlation diagram. The lags between spikes of a test and a reference trains are plotted against the time of occurrence or the rank of the reference train spikes.

Usage

```
lockedTrain(stRef, stTest, laglim, acquisitionWindow)
## S3 method for class 'lockedTrain':
plot(x, keepTime = FALSE,
             stimTimeCourse = NULL, colStim = "grey80",
             xlim, pch, xlab, ylab, main, ...)
## S3 method for class 'lockedTrain':
print(x,...)
```

Arguments

`stRef`	a `spikeTrain` or a `repeatedTrain` object.
`stTest`	a `spikeTrain` or a `repeatedTrain` object. If `missing(stTest)` is `TRUE` then `stRef` is used.
`x`	a `lockedTrain` object.
`laglim`	a two elements vector, the time window (in s) in which spikes in `stTest` around spikes in `stRef` are looked for. Default value are supplied when the argument is missing (+/- 3 times the sd of the inter-spike intervals of `stRef`).
`acquisitionWindow`	a 2 elements vector specifying the begining and the end of the acquisition. If `missing` values are obtained using the `floor` of the smallest spike time and the `ceiling` of the largest one.
`keepTime`	a logical, if `TRUE` the ordinate is shown in s, otherwise (default) the spike index is shown.

`stimTimeCourse`	`NULL` (default) or a two elements vector specifying the time boundaries (in s) of a stimulus presentation.
`colStim`	the background color used for the stimulus.
`xlim`	a numeric (default value supplied). See `plot`.
`pch`	data symbol used for the spikes. See `plot`.
`xlab`	a character (default value supplied). See `plot`.
`ylab`	a character (default value supplied). See `plot`.
`main`	a character (default value supplied). See `plot`.
`...`	see `plot` or `print`.

Details

The time-dependent cross-correlation diagram is described in van Stokkum et al (1986) and is also used by Brillinger (1992) Fig. 4. For each spike of `stRef` neighboring spikes of `stTest` are selected within a window defined by `laglim`. The lag between these `stTest` spikes and the ones of `stRef` are displayed (that is, the times of the `stRef` spikes is subtracted from the times of the neighboring spikes in `stTest`).

If `repeatedTrains` are given for `stRef` and `stTest` they must have the same number of components and are interpreted as coming from repetitions of the same stimulation, the spike times of the different trains of `stRef` are therefore reordered.

The ordinate on the plot generated by `plot.lockedTrain` can be in term of real time or in term of `stRef` spike indexes.

If `stimTimeCourse` is specified a box corresponding to the stimulus presentation is drawn in the background.

Value

`lockedTrain` returns a LIST of class `lockedTrain` with the following components:

`shiftedT`	a list of lists. Each sublist has three components: `refTime`, the time of the reference spike; `repIdx`, the index of the stimulus repeat to which the reference spike belongs; `crossTime`, a vector of shifted times of the test neurons. These times are shifted because they are expressed with respect to the reference spike time.
`nbRefSpikes`	the total number of reference spikes used.
`nbTestSpikes`	the total number of test spikes occurring during the same observation period.
`laglim`	the value of `laglim` used.
`acquisitionWindow`	the value of the `acquisitionWindow` used.
`obsTime`	the total observation time used (in s).
`call`	the matched call.

`plot.lockedTrain` and `print.lockedTrain` are used for their side effects: a plot is generated. `print.lockedTrain` calls `plot.lockedTrain`.

Note

`plot.lockedTrain` displays essentially the "raw data" from which a cross-intensity histogram is built.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

van Stokkum, I. H., Johannesma, P. I. and Eggermont, J. J. (1986) Representation of time-dependent correlation and recurrence time functions. A new method to analyse non-stationary point processes. *Biol Cybern* **55**: 17–24.

Brillinger, David R. (1992) Nerve Cell Spike Train Data Analysis: A Progression of Technique. *JASA* **87**: 260–271.

See Also

`as.spikeTrain`, `as.repeatedTrain`, `raster`

Examples

```
## Not run:
## load spontaneous data of 4 putative projection neurons
## simultaneously recorded from the cockroach (Periplaneta
## americana) antennal lobe
data(CAL1S)
## convert data into spikeTrain objects
CAL1S <- lapply(CAL1S,as.spikeTrain)
## look at the individual trains
## first the "raw" data
CAL1S[["neuron 1"]]
## contruct the lockedTrain of each neuron with itself and look at
## it using a lag of +/- 25 ms
lockedTrain(CAL1S[["neuron 1"]],laglim=c(-1,1)*0.025)
lockedTrain(CAL1S[["neuron 2"]],laglim=c(-1,1)*0.025)
lockedTrain(CAL1S[["neuron 3"]],laglim=c(-1,1)*0.025)
lockedTrain(CAL1S[["neuron 4"]],laglim=c(-1,1)*0.025)

## Look at the Vanillin responses
## Get the data
data(CAL1V)
## convert them into repeatedTrain objects
## The stimulus command is on between 4.49 s and 4.99s
CAL1V <- lapply(CAL1V,as.repeatedTrain)
## look at the individual raster plots
plot(CAL1V[["neuron 1"]],stimTimeCourse=c(4.49,4.99),main="N1")
plot(CAL1V[["neuron 2"]],stimTimeCourse=c(4.49,4.99),main="N2")
plot(CAL1V[["neuron 3"]],stimTimeCourse=c(4.49,4.99),main="N3")
plot(CAL1V[["neuron 4"]],stimTimeCourse=c(4.49,4.99),main="N4")
## construct the locked train for the 3 pairs with neuron 1 as a
## reference
plot(lockedTrain(CAL1V[["neuron 1"]],CAL1V[["neuron 3"]],
     laglim=0.01*c(-1,1)),stimTimeCourse=c(4.49,4.99),pch="*")
plot(lockedTrain(CAL1V[["neuron 1"]],CAL1V[["neuron 2"]],
     laglim=0.01*c(-1,1)),stimTimeCourse=c(4.49,4.99),pch="*")
plot(lockedTrain(CAL1V[["neuron 1"]],CAL1V[["neuron 4"]],
     laglim=0.01*c(-1,1)),stimTimeCourse=c(4.49,4.99),pch="*")
## End(Not run)
```

mkDummy — *Generates a Data Frame of Dummy Variables for Use in gam*

Description

Using argument `by` in `s` or `te` of `gam` requires dummy variables to be set up. This is the job of this function.

Usage

```
mkDummy(x)
```

Arguments

`x`	a `factor`.

Value

A `data.frame` with as many variables as there are `levels` in `x` and as many rows as elements in `x`.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`mkGLMdf`, `gam`, `s`, `te`

Examples

```
## coming soon
```

mkGLMdf — *Formats (lists of) spikeTrain and repeatedTrain Objects into Data Frame for use in glm, mgcv and gam*

Description

Given a `spikeTrain` or a `repeatedTrain` objects or a `list` of any of those two, `mkGLMdf` generates a `data.frame`, by discretizing time, allowing `glm` and `gam` to be used with the `poisson` or `binomial` family to fit the spike trains.

Usage

```
mkGLMdf(obj, delta, lwr, upr, discrete = TRUE)
```

Arguments

`obj`	a `spikeTrain` or a `repeatedTrain` objects or a `list` of any of those two.
`delta`	the bin size used for time discretization (in s).
`lwr`	the time (in s) at which the recording window starts. If `missing` a value is obtained using the `floor` of the smallest spike time.
`upr`	the time (in s) at which the recording window ends. If `missing` a value is obtained using the `ceiling` of the largest spike time.
`discrete`	a logical. Should time differences be reported in bin (default value) or as actual times (`FALSE`)?

Details

If `discrete` is set to `FALSE` the actual time differences (from the "raw" data in `obj`) are reported.

The construction of the returned list is very clearly explained in Jim Lindsey's paper (1995). The idea has been used several time in the field: Brillinger (1988), Kass and Ventura (2001), Truccolo et al (2005).

Value

A `data.frame` with the following variables:

`event`	an integer presence (1) or absence (0) of an event from a given neuron in the given bin.
`time`	time at bin center.
`neuron`	a factor giving the neuron to which this row of the data frame refers.
`lN.x`	an integer (if `discrete` is `TRUE`) or a numeric (if `discrete` is `FALSE`). `x` takes value 1, 2, ..., number of neurons present in `obj`. The time to the last event of the corresponding neuron.

The list has also few attributes: `lwr`, the start of the recording window; `upr`, the end of the recording window; `delta`, the bin width; `call`, the call used to generate the list.

Note

See the example bellow to get an idea of what to do with the returned list.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Lindsey, J. K. (1995) Fitting Parametric Counting Processes by Using Log-Linear Models *Applied Statistics* **44**: 201–212.

Brillinger, D. R. (1988) Maximum likelihood analysis of spike trains of interacting nerve cells *Biol Cybern* **59**: 189–200.

Kass, Robert E. and Ventura, Valérie (2001) A spike-train probability model *Neural Comput.* **13**: 1713–1720.

Truccolo, W., Eden, U. T., Fellows, M. R., Donoghue, J. P. and Brown, E. N. (2005) A Point Process Framework for Relating Neural Spiking Activity to Spiking History, Neural Ensemble and Extrinsic Covariate Effects *J Neurophysiol* **93**: 1074–1089. `http://jn.physiology.org/cgi/content/abstract/93/2/1074`

See Also

data.frame, glm, mgcv, as.spikeTrain, as.repeatedTrain

Examples

```
## Not run:
## Start with simulatd data #####
## Use thinning method and for that define a couple
## of functions

## expDecay gives an exponentially decaying
## synaptic effect followin a presynpatic spike.
## All the pre-synaptic spikes between "now" (argument
## t) and the previous spike of the post-synaptic
## neuron have an effect (and the summation is linear)
expDecay <- function(t,preT,last,
                     delay=0.002,tau=0.015) {

  if (missing(last)) good <- (preT+delay) < t
  else good <- last < preT & (preT+delay) < t
  if (sum(good) == 0) return(0)
  preS <- preT[good]
  preS <- t-preS-delay
  sum(exp(-preS/tau))

}

## Same as expDecay except that the effect is pusle like
pulseFF <- function(t,preT,last,
                    delay=0.005,duration=0.01) {
  if (missing(last)) good <- t-duration < (preT+delay) & (preT+delay) < t
  else good <- t-duration < (preT+delay) & last < preT & (preT+delay) < t
  sum(good)
}

## The work horse. Given a pre-synaptic train (preT),
## a duration, lognormal parameters and a presynaptic
## effect fucntion, mkPostTrain simulates a log-linear
## post-synaptic train using the thinning method
mkPostTrain <- function(preT,
                        duration=60,
                        meanlog=-2.4,
                        sdlog=0.4,
                        preFF=expDecay,
                        beta=log(5),
                        maxCI=30,
                        ...) {

  nuRest <- exp(-meanlog-0.5*sdlog^2)
  poissonRest <- nuRest*ifelse(beta>0,exp(beta),1)
  ciRest <- function(t) nuRest*exp(beta*preFF(t,preT,...))

  poissonNext <- maxCI*ifelse(beta>0,exp(beta),1)
  ci <- function(t,tLast) hlnorm(t-tLast,meanlog,sdlog)*exp(beta*preFF(t,preT,tLast,...))

  vLength <- poissonRest*300
  result <- numeric(vLength)
  currentTime <- 0
  lastTime <- 0
  eventIdx <- 1

  nextTime <- function(currentTime,lastTime) {
    if (currentTime > 0) {
      currentTime <- currentTime + rexp(1,poissonNext)
      ciRatio <- ci(currentTime,lastTime)/poissonNext
      if (ciRatio > 1) stop("Problem with thinning.")
      while (runif(1) > ciRatio) {
        currentTime <- currentTime + rexp(1,poissonNext)
        ciRatio <- ci(currentTime,lastTime)/poissonNext
        if (ciRatio > 1) stop("Problem with thinning.")
      }
    } else {
      currentTime <- currentTime + rexp(1,poissonRest)
      ciRatio <- ciRest(currentTime)/poissonRest
      if (ciRatio > 1) stop("Problem with thinning.")
      while (runif(1) > ciRatio) {
        currentTime <- currentTime + rexp(1,poissonRest)
        ciRatio <- ciRest(currentTime)/poissonRest
        if (ciRatio > 1) stop("Problem with thinning.")
      }
    }
    currentTime
  }

  while(currentTime <= duration) {
    currentTime <- nextTime(currentTime,lastTime)
    result[eventIdx] <- currentTime
    lastTime <- currentTime
    eventIdx <- eventIdx+1
    if (eventIdx > vLength) {
      result <- c(result,numeric(vLength))
      vLength <- length(result)
    }
  }
  result[result > 0]

}

## set the rng seed
set.seed(11006,"Mersenne-Twister")
## generate a log-normal pre train
preTrain <- cumsum(rlnorm(1000,-2.4,0.4))
preTrain <- preTrain[preTrain < 60]
## generate a post synaptic train with an
## exponentially decaying pre-synaptic excitation
post1 <- mkPostTrain(preTrain)
## generate a post synaptic train with a
## pulse-like pre-synaptic excitation
post2 <- mkPostTrain(preTrain,preFF=pulseFF)
## generate a post synaptic train with a
## pulse-like pre-synaptic inhibition
post3 <- mkPostTrain(preTrain,preFF=pulseFF,beta=-log(5))
## make a list of spikeTrain objects out of that
```

```
interData <- list(pre=as.spikeTrain(preTrain),
                  post1=as.spikeTrain(post1),
                  post2=as.spikeTrain(post2),
                  post3=as.spikeTrain(post3))
## remove the trains
rm(preTrain,post1,post2,post3)
## look at them
interData[["pre"]]
interData[["post1"]]
interData[["post2"]]
interData[["post3"]]
## compute cross-correlograms
interData.lt1 <- lockedTrain(interData[["pre"]],interData[["post1"]],laglim=c(-0.03,0.05)
interData.lt2 <- lockedTrain(interData[["pre"]],interData[["post2"]],laglim=c(-0.03,0.05)
interData.lt3 <- lockedTrain(interData[["pre"]],interData[["post3"]],laglim=c(-0.03,0.05)
## look at the cross-raster plots
interData.lt1
interData.lt2
interData.lt3
## look at the corresponding histograms
hist(interData.lt1,bw=0.0025)
hist(interData.lt2,bw=0.0025)
hist(interData.lt3,bw=0.0025)
## check out what goes on between post2 and post1
interData.lt1v2 <- lockedTrain(interData[["post2"]],interData[["post1"]],laglim=c(-0.03,0
interData.lt1v2
hist(interData.lt1v2,bw=0.0025)

## fine
## create a GLM data frame using a 1 ms bin width
dfAll <- mkGLMdf(interData,delta=0.001,lwr=0,upr=60)
## build the sub-list relating to neuron 2
dfN2 <- dfAll[dfAll$neuron=="2",]
## load the mgcv library
library(mgcv)
## fit dfN2 with a smooth effect for the elasped time since the last
## event of neuron 2 and another one with the elasped time since the
## last event from neuron 1. Use moroever only the events for which the
## the last event from neuron 1 occurred at most 100 ms ago.
dfN2.fit0 <- gam(event ~ s(lN.1,bs="cr") + s(lN.2,bs="cr"), data=dfN2, family=poisson, su
## look at the summary
summary(dfN2.fit0)
## plot the smooth term of neuron 1
plot(dfN2.fit0,select=1,rug=FALSE,ylim=c(-0.8,0.8))
## Can you see the exponential presynatic effect with
## a 15 ms decay time appearing?
## Now check the dependence on lN.2
xx <- seq(0.001,0.3,0.001)
## plot the estimated conditional intensity when the last spike
## from neuron 1 came a long time ago (100 ms)
plot(xx,exp(predict(dfN2.fit0,data.frame(lN.1=rep(100,300),lN.2=1:300))),type="l")
## add a line for the true conditional intensity
lines(xx,hlnorm(xx,-2.4,0.4)*0.001,col=2)
## do the same thing for the survival function
plot(xx,exp(-cumsum(exp(predict(dfN2.fit0,data.frame(lN.1=rep(100,300),lN.2=1:300))))),ty
lines(xx,plnorm(xx,-2.4,0.4,lower.tail=FALSE),col=2)
```

```
## Now repeat the fit including a possible contribution from neuron 3
dfN2.fit1 <- gam(event ~ s(lN.1,bs="cr") + s(lN.2,bs="cr") + s(lN.3,bs="cr"), data=dfN2,
## Use the summary to see if the new element brings something
summary(dfN2.fit1)
## It does not!
## Now look at neurons 3 and 4 (ie, post2 and post3)
dfN3 <- dfAll[dfAll$neuron=="3",]
dfN3.fit0 <- gam(event ~ s(lN.1,k=20,bs="cr") + s(lN.3,k=15,bs="cr"),data=dfN3,family=poi
summary(dfN3.fit0)
plot(dfN3.fit0,select=1,ylim=c(-1.5,1.8),rug=FALSE)
dfN4 <- dfAll[dfAll$neuron=="4",]
dfN4.fit0 <- gam(event ~ s(lN.1,k=20,bs="cr") + s(lN.4,k=15,bs="cr"),data=dfN4,family=poi
summary(dfN4.fit0)
plot(dfN4.fit0,select=1,ylim=c(-1.8,1.5),rug=FALSE)
## End(Not run)
```

mkREdf *Evaluates RateEvolutions for spikeTrain Lists and Returns Data Frame*

Description

Given a list of `spikeTrain` or `repeatedTrain` objects `mkREdf` evaluates the rate evolution of each train and returns a data frame suitable for use with `coplot`, `xyplot` and `qplot`.

Usage

```
mkREdf(x, longitudinal, across, bw,
       kernel=c("gaussian", "epanechnikov", "rectangular",
                "triangular", "biweight", "cosine", "optcosine"),
       n=512, from, to, na.rm=FALSE, minusMean=FALSE)
```

Arguments

`x`	a *named* list of `spikeTrain` or `repeatedTrain` objects.
`longitudinal`	a `character` vector with the names of the different "conditions" applied to each neuron like "ctl", "bicu" or "stim. 1", "stim. 2", ..., "stim. 20". Default provided.
`across`	a `character` vector with the names of the different neurons. Default provided.
`bw`	see `rateEvolution`. This can be a vector.
`kernel`	see `rateEvolution`.
`n`	see `rateEvolution`.
`from`	see `rateEvolution`.
`to`	see `rateEvolution`.
`na.rm`	see `rateEvolution`.
`minusMean`	should the mean of the rate evolution along the across "dimension" be subtracted from each individual rate evolution along this dimension?

Details

`mkREdf` calls `rateEvolution` on every `spikeTrain` in `x`. If `from` and `to` are missing, they are internally set to the `floor` of the global minimal spike time contained in `x` and to the `ceiling` of the global maximal time.

Value

A data frame with the following variables:

`time`	The time (in s) at which the rate was evaluated.
`rate`	The rate (in 1/s).
`longitudinal`	A factor corresponding to the argument with the same name.
`across`	A factor corresponding to the argument with the same name.

Note

argument `minusMean` is now here as an "experimental" feature. The idea is that it could be used to detect non-stationarities of the reponses (in a repeated stimulation context) which would be correlated across different neurons. I'm not sure yet if this will be useful or not.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`as.spikeTrain`, `as.repeatedTrain`, `data.frame`, `factor`, `rateEvolution`,

Examples

```
## load Purkinje cell data recorded in cell-attached mode
data(sPK)
## coerce sPK to a spikeTrain object
sPK <- lapply(sPK, as.spikeTrain)
## get a rate evolution data frame
sPKreDF <- mkREdf(sPK)
## display result using coplot
coplot(rate ~ time | longitudinal,data=sPKreDF,panel=lines,show.given=FALSE)
## Not run:
## make it prettier with with xyplot of package lattice
library(lattice)
xyplot(rate ~ time | longitudinal, data=sPKreDF,panel=panel.lines)
## if ggplot2 is installed, try it out
library(ggplot2)
qplot(time,rate,data=sPKreDF,geom="line",colour=longitudinal)
## End(Not run)

## load Purkinje cell data recorded with the NeuroNexus probes
data(mPK)
mPK <- lapply(mPK, as.repeatedTrain)
## get a rate evolution data frame
mPKreDF <- mkREdf(mPK)
## use coplot to display result
coplot(rate ~ time | longitudinal * across,data = mPKreDF,panel=lines)
```

```
## Not run:
## make it prettier with with xyplot of package lattice
library(lattice)
xyplot(rate ~ time | across,data = mPKreDF,groups=longitudinal,panel=panel.lines)
xyplot(rate ~ time | across * longitudinal,data = mPKreDF, panel=panel.lines)
## if ggplot2 is installed, try it out
library(ggplot2)
qplot(time,rate,data=mPKreDF,geom="line",colour=longitudinal,facets=across ~ .)
## End(Not run)

## another example with the CAL1V data set
data(CAL1V)
CAL1V <- lapply(CAL1V,as.repeatedTrain)
## generate the data frame specifying the longitudinal argument
## to end up with a clearer display
CAL1VreDF <- mkREdf(CAL1V,longitudinal=paste(1:20))
coplot(rate ~ time | across * longitudinal,data=CAL1VreDF,panel=lines,show.given=FALSE)
## Not run:
## if ggplot2 is installed, try it out
library(ggplot2)
qplot(time,rate,data=CAL1VreDF,geom="line",facets=longitudinal ~ across)
## End(Not run)

## another example with the CAL2C data set
data(CAL2C)
CAL2C <- lapply(CAL2C,as.repeatedTrain)
## generate the data frame specifying the longitudinal argument
## to end up with a clearer display
CAL2CreDF <- mkREdf(CAL2C,longitudinal=paste(1:20))
coplot(rate ~ time | across * longitudinal,data=CAL2CreDF,panel=lines,show.given=FALSE)
## Not run:
## if ggplot2 is installed, try it out
library(ggplot2)
qplot(time,rate,data=CAL2CreDF,geom="line",facets=longitudinal ~ across)
## End(Not run)
```

`plot.frt` *Plots and Summarizes frt Objects.*

Description

`plot.frt` generates interactively (by default) 2 plots, the survivor function with confidence intervals and the Berman's test with confidence bands. `summary.frt` generates a concise summary of `frt` objects. It is mostly intended for use in batch processing situations where a decision to stop with the current model or go on with a more complicated one must be made automatically.

Usage

```
## S3 method for class 'frt':
plot(x, which = 1:2, main,
          caption = c("Log Survivor Function", "Berman's Test"),
          ask = TRUE, ...)
## S3 method for class 'frt':
summary(object, ...)
```

Arguments

`x`	a `transformedTrain` object.
`object`	a `transformedTrain` object.
`which`	if a subset of the plots is required, specify a subset of the numbers `1:2`.
`main`	title to appear above the plots, if missing the corresponding element of `caption` will be used.
`caption`	Default caption to appear above the plots or, if `main` is given, bellow it
`ask`	logical; if `TRUE`, the user is *ask*ed before each plot, see `par(ask=.)`.
`...`	additional arguments passed to `plot`.

Details

If the reference and test (transformed) spike trains used in the `frt` call which generated `x` (or `object`) are not correlated (and if the transformed test train is indeed homogeneous Poisson with rate 1), the elements of `x` (or `object`) should be iid realizations of an exponential with rate 1. Two test plots are generated by `plot.frt` in the same way as the corresponding ones (testing the same thing) of `plot.transformedTrain`.

The same correspondence holds between `summary.frt` and `summary.transformedTrain`.

Value

`summary.frt` returns a vector with named elements stating if the Berman's test is passed with a 95% and a 99% confidence.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`transformedTrain`, `frt`, `mkGLMdf`

Examples

```
## Not run:
## Let us consider neuron 1 of the CAL2S data set
data(CAL2S)
CAL2S <- lapply(CAL2S,as.spikeTrain)
CAL2S[["neuron 1"]]
renewalTestPlot(CAL2S[["neuron 1"]])
summary(CAL2S[["neuron 1"]])
## Make a data frame with a 4 ms time resolution
cal2Sdf <- mkGLMdf(CAL2S,0.004,0,60)
## keep the part relative to neuron 1, 2 and 3 separately
n1.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="1",]
n2.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="2",]
n3.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="3",]
## remove unnecessary data
rm(cal2Sdf)
## Extract the elapsed time since the second to last and
## third to last for neuron 1. Normalise the result.
n1.cal2sDF[c("rlN.1","rsN.1","rtN.1")] <- brt4df(n1.cal2sDF,"lN.1",2,c("rlN.1","rsN.1","r
## load mgcv library
library(mgcv)
## fit a model with a tensorial product involving the last
## three spikes and using a cubic spline basis for the last two
## To gain time use a fixed df regression spline
n1S.fitA <- gam(event ~ te(rlN.1,rsN.1,bs="cr",fx=TRUE) + rtN.1,data=n1.cal2sDF,family=bi
## transform time
N1.Lambda <- transformedTrain(n1S.fitA)
## check out the resulting spike train using the fact
## that transformedTrain objects inherit from spikeTrain
## objects
N1.Lambda
## Use more formal checks
summary(N1.Lambda)
plot(N1.Lambda,which=c(1,2,4,5),ask=FALSE)
## Transform spike trains of neuron 2 and 3
N2.Lambda <- transformedTrain(n1S.fitA,n2.cal2sDF$event)
N3.Lambda <- transformedTrain(n1S.fitA,n3.cal2sDF$event)
## Check interactions
summary(N2.Lambda %frt% N1.Lambda)
summary(N3.Lambda %frt% N1.Lambda)
plot(N2.Lambda %frt% N1.Lambda,ask=FALSE)
plot(N3.Lambda %frt% N1.Lambda,ask=FALSE)
## End(Not run)
```

plot.spikeTrain *Display Counting Process Associated with Single Spike Train*

Description

Adds a counting process display to the classical raster plot of single spike trains.

Usage

```
## S3 method for class 'spikeTrain':
plot(x, xlab = "Time (s)", ylab = "Cumulative Number of Events",
                  main = paste("Counting Process of",deparse(substitute(x))),
                  xlim = c(floor(x[1]), ceiling(x[length(x)])),
                  ylim = c(0, length(x) + 1),
                  do.points = ifelse(length(x) < 100, TRUE, FALSE),
                  addMeanRate = TRUE, addRug = TRUE, ...)
```

Arguments

`x`	a `spikeTrain` object or a vector which can be coerced to such an object.
`xlab`	a character. The x label.
`ylab`	a character. The y label.
`main`	a character. The title.
`xlim`	a numeric. See `plot`.
`ylim`	a numeric. See `plot`.
`do.points`	see `plot.stepfun`.

`addMeanRate`	should the expected counting process for a Poisson process with the same rate be added to the plot?
`addRug`	should a rug representation be added at teh bottom of the plot? See `rug`.
`...`	additional arguments passed to `plot`, see `plot` and `plot.stepfun`.

Details

The counting process is obtained by a call to `stepfun`. When `xlab`, `ylab`, `main`, `xlim` or `ylim` is (are) missing, default values are used.

Value

Nothing is returned, `plot.spikeTrain` is used for its side effect, a plot is generated on the current graphic device.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

D. R. Cox and P. A. W. Lewis (1966) *The Statistical Analysis of Series of Events*. John Wiley and Sons.

Brillinger, D. R. (1988) Maximum likelihood analysis of spike trains of interacting nerve cells. *Biol. Cybern.* **59**: 189–200.

Johnson, D.H. (1996) Point process models of single-neuron discharges. *J. Computational Neuroscience* **3**: 275–299.

See Also

`as.spikeTrain`, `is.spikeTrain`, `print.spikeTrain`, `summary.spikeTrain`, `renewalTestPlot`, `varianceTime`, `stepfun`, `plot.stepfun`, `rug`

Examples

```
## Not run:
data(ShallowShocks)
plot(as.spikeTrain(ShallowShocks$Date),
     xlab="Time (days)",
     main="Shallow Shocks Counting Process of Ogata 1988")
## End(Not run)
```

`plot.transformedTrain` *Plot Diagnostics for an transformedTrain Object*

Description

Six plots (selectable by `which`) are currently available: the first 5 of which correspond to Fig. 9 to 13 of Ogata (1988). The sixth one is new (as far as I know) and is still "experimental". They are all testing the first argument of `plot.transformedTrain` against the Poisson process hypothesis..

Usage

```
## S3 method for class 'transformedTrain':
plot(x, which = 1:5, main,
                      caption = c("Uniform on Trans. Obs. Time Test",
                                  "Berman's Test",
                                  "Log Survivor Function",
                                  "Lag 1 Transformed Intervals",
                                  "Variance vs Mean",
                                  "Martingale vs Trans. Time"),
                      ask = TRUE,
                      ...)
```

Arguments

`x`	a `transformedTrain` object.
`which`	if a subset of the plots is required, specify a subset of the numbers `1:6`.
`main`	title to appear above the plots, if missing the corresponding element of `caption` will be used.
`caption`	Default caption to appear above the plots or, if `main` is given, bellow it
`ask`	logical; if `TRUE`, the user is *ask*ed before each plot, see `par(ask=.)`.
`...`	not used only there for compatibility with `plot` generic method.

Details

If the `transformedTrain` object `x` is a the realization of a homogeneous Poisson process then, conditioned on the number of events observed, the location of the events is uniform on the (time transformed) period of observation. This is a basic property of the homogeneous Poisson process derived in Chap. 2 of Cox and Lewis (1966) and Daley and Vere-Jones (2003). This is what the first plot generated (by default) tests with a Kolmogorov-Smirnov Test. The two dotted lines on both sides of the diagonal correspond to 95 and 99% confidence intervals. This is the plot shown on Fig. 9 (p 19) of Ogata (1988).

If we write x_i the elements of the `transformedTrain` object `x` and if the latter is the realization of a homogeneous Poisson process then the intervals:

$$y_i = x_{i+1} - x_i$$

are iid rv from an exponential distribution with rate 1 and the:

$$u_i = 1 - \exp(-y_i$$

are iid rv from a uniform distribution on [0,1). The second plot generated (by default) tests this uniform distribution hypotheses with a Kolmogorov-Smirnov Test. This is the plot shown on Fig. 10 (p 19) of Ogata (1988) which was suggested by Berman. This is also the plot proposed by Brown et al (2002). The two dotted lines on both sides of the diagonal correspond to 95 and 99% confidence intervals.

Following the line of the previous paragraph, if the distribution of the y_i is an exponential distribution with rate 1, then their survivor function is: $\exp(-y)$. This is what's shown on the third plot generated (by default) using a log scale for the ordinate. The point wise CI at 95 and 99% are also drawn (dotted lines). This is the plot shown on Fig. 12 (p 20) of Ogata (1988)

If the u_i of the second paragraph are iid uniform rv on [0,1) then a plot of u_{i+1} vs u_i should fill uniformly the unit square [0,1) x [0,1). This is the fourth generated plot (by default). This is the plot shown on Fig. 11 (p 20) of Ogata (1988)

If the x_i are realization of a homogeneous Poisson process observed between 0 and T (on the transformed time scale), then the number of events observed on non-overlapping windows of length t should be iid Poisson rv with mean t (and variance t). The observation period is chopped into non-overlapping windows of increasing length and the empirical variance of the event count is plotted versus the empirical mean, together with 95 and 99% CI (using a normal approximation). This is done by calling internally `varianceTime`. That's what's generated by the fifth plot (by default). This is the plot shown on Fig. 13 (p 20) of Ogata (1988)

The last plot is experimental and irrelevant for spike trains transformed after a `gam` or a `glm` fit. It should be useful for parametric models fitted with the maximum likelihood method.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Cox, D. R. and Lewis, P. A. W. (1966) *The Statistical Analysis of Series of Events*. John Wiley and Sons.

Daley, D. J. and Vere-Jones D. (2003) *An Introduction to the Theory of Point Processes. Vol. 1*. Springer.

Ogata, Yosihiko (1988) Statistical Models for Earthquake Occurrences and Residual Analysis for Point Processes. *Journal of the American Statistical Association* **83**: 9-27.

Brown, E. N., Barbieri, R., Ventura, V., Kass, R. E. and Frank, L. M. (2002) The time-rescaling theorem and its application to neural spike train data analysis. *Neural Computation* **14**: 325-346.

See Also

`transformedTrain`, `summary.transformedTrain`, `mkGLMdf`

Examples

```
## Not run:
## Let us consider neuron 1 of the CAL2S data set
data(CAL2S)
CAL2S <- lapply(CAL2S,as.spikeTrain)
CAL2S[["neuron 1"]]
renewalTestPlot(CAL2S[["neuron 1"]])
summary(CAL2S[["neuron 1"]])
## Make a data frame with a 4 ms time resolution
cal2Sdf <- mkGLMdf(CAL2S,0.004,0,60)
## keep the part relative to neuron 1, 2 and 3 separately
n1.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="1",]
n2.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="2",]
n3.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="3",]
## remove unnecessary data
rm(cal2Sdf)
## Extract the elapsed time since the second to last and
## third to last for neuron 1. Normalise the result.
n1.cal2sDF[c("rlN.1","rsN.1","rtN.1")] <- brt4df(n1.cal2sDF,"lN.1",2,c("rlN.1","rsN.1","r
## load mgcv library
library(mgcv)
## fit a model with a tensorial product involving the last
## three spikes and using a cubic spline basis for the last two
## To gain time use a fixed df regression spline
```

```
n1S.fitA <- gam(event ~ te(rlN.1,rsN.1,bs="cr",fx=TRUE) + rtN.1,data=n1.cal2sDF,family=bi
## transform time
N1.Lambda <- transformedTrain(n1S.fitA)
## check out the resulting spike train using the fact
## that transformedTrain objects inherit from spikeTrain
## objects
N1.Lambda
## Use more formal checks
summary(N1.Lambda)
plot(N1.Lambda,which=c(1,2,4,5),ask=FALSE)
## Transform spike trains of neuron 2 and 3
N2.Lambda <- transformedTrain(n1S.fitA,n2.cal2sDF$event)
N3.Lambda <- transformedTrain(n1S.fitA,n3.cal2sDF$event)
## Check interactions
summary(N2.Lambda %frt% N1.Lambda)
summary(N3.Lambda %frt% N1.Lambda)
plot(N2.Lambda %frt% N1.Lambda,ask=FALSE)
plot(N3.Lambda %frt% N1.Lambda,ask=FALSE)
## End(Not run)
```

---

print.repeatedTrain *Print and Summary Methods for repeatedTrain Objects*

Description

Print and summary `methods` for `repeatedTrain` objects.

Usage

```
## S3 method for class 'repeatedTrain':
print(x,...)
## S3 method for class 'repeatedTrain':
summary(object,
                        responseWindow, acquisitionWindow,...)
## S3 method for class 'summary.repeatedTrain':
print(x,...)
```

Arguments

`x`	a `repeatedTrain` or a `summary.repeatedTrain` object.
`object`	a `repeatedTrain` object
`responseWindow`	a 2 elements vector specifying the begining and the end of the neuron response.
`acquisitionWindow`	a 2 elements vector specifying the begining and the end of the acquisition. If `missing` values are obtained using the `floor` of the smallest spike time and the `ceiling` of the largest one.
`...`	additional arguments passed to function `chisq.test` or `print`.

Details

`print.repeatedTrain` calls `plot.repeatedTrain`

Value

`summary.repeatedTrain` returns a LIST of class `summary.repeatedTrain` with the following components:

`nbRepeates`	The number of repetitions.
`acquisitionWindow`	The acquisition window.
`stats`	A matrix with as many rows as repetitions. The first column contains the total number of spikes generated by the neuron during a given repeat (this column appears under the heading "nb" when the object is printed). The second column contains the corresponding average discharge rate (this column appears under the heading "nu" when the object is printed). If a `responseWindow` was specified, the third column contains the number of spikes generated by the neuron during the response period and the fourth column contains the corresponding rate (these column appear under the headings "nbR" and "nuR", respectively when the object is printed).
`globalPval`	The p value of the chi square test for homogeneity of the total number of spikes generated accross repetitions. Thats a rough stationarity test.
`responsePval`	If a `responseWindow` was specified, the p value of the chi square test for homogeneity of the number of spikes generated within the "response window" accross repetitions.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`as.repeatedTrain`, `is.repeatedTrain`, `plot.repeatedTrain`, `raster`, `psth`

Examples

```
## Load the Vanillin responses of the first
## cockroach data set
data(CAL1V)
## convert them into repeatedTrain objects
## The stimulus command is on between 4.49 s and 4.99s
CAL1V <- lapply(CAL1V,as.repeatedTrain)
## Generate raster plot for the neurons
raster(CAL1V[["neuron 1"]],c(4.49,4.99))
plot(CAL1V[["neuron 2"]],c(4.49,4.99))
plot(CAL1V[["neuron 3"]],c(4.49,4.99))
## Basic summary of neuron 1
summary(CAL1V[["neuron 1"]])
## Enhanced summary giving a response window between 5 and 5.5s
summary(CAL1V[["neuron 1"]],c(5,5.5))
```

`print.spikeTrain` *Print and Summary Methods for spikeTrain Objects*

Description

Print and summary `methods` for `spikeTrain` objects.

Usage

```
## S3 method for class 'spikeTrain':
print(x,...)
## S3 method for class 'spikeTrain':
summary(object, timeUnit = "s", digits = 3, ...)
```

Arguments

`x, object`	A `spikeTrain` object.
`timeUnit`	The unit with which the occurrence times were measured.
`digits`	The number of digits used to print the summary (see `round`).
`...`	see `print` and `summary`.

Details

`print.spikeTrain` does in fact call the `plot` method for `spikeTrain` objects.

Value

`print.spikeTrain` generates a plot as a side effect.

`summary.spikeTrain` returns the number of spikes, the times of the first and last spikes, the mean inter-spike interval (ISI) and its sd as well as the mean and sd of the log(ISI) together with the shortest and longest ISIs.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`as.spikeTrain`, `is.spikeTrain`, `renewalTestPlot`, `varianceTime`, `stepfun`

Examples

```
## load spontaneous data of 4 putative projection neurons
## simultaneously recorded from the cockroach (Periplaneta
## americana) antennal lobe
data(CAL1S)
## convert data into spikeTrain objects
CAL1S <- lapply(CAL1S,as.spikeTrain)
## look at the individual trains
## first the "raw" data
CAL1S[["neuron 1"]]
## next some summary information
summary(CAL1S[["neuron 1"]])
```

psth *Compute and Plot Peri-Stimulus Time Histogram*

Description

`psth` computes and `plot.psth` plots a peri-stimulus time histogram (called PST, post-stimulus time histogram by Gerstein and Kiang (1960)) from repeated presentations of a stimulation. Confidence bands can be obtained using the Poisson approximation.

Usage

```
psth(repeatedTrain, breaks = 20, include.lowest = TRUE,
     right = TRUE, plot = TRUE, CI = 0.95, ...)
## S3 method for class 'psth':
plot(x, stimTimeCourse = NULL, colStim = "grey80",
           colCI = NULL, xlab, ylab, main, xlim, ylim, lwd = 2,
           col = 1, ...)
```

Arguments

`repeatedTrain`	a `repeatedTrain` object or a list which can be coerced to such an object.
`x`	a `psth` object.
`stimTimeCourse`	`NULL` (default) or a two elements vector specifying the time boundaries (in s) of a stimulus presentation.
`colStim`	the background color used for the stimulus.
`breaks`	a numeric. A single number is interpreted has the number of bins; a vector of length 2 is interpreted as the bin width and the step to use (see details); otherwise interpreted as the position of the "breaks" between bins.
`include.lowest`	corresponding argument of `hist`.
`right`	corresponding argument of `hist`.
`plot`	corresponding argument of `hist`.
`CI`	The coverage probability of the confidence intervals.
`colCI`	if not `NULL` (default) a confidence band is plotted with the specified color; two dashed lines are plotted otherwise.
`xlim`	a numeric (default value supplied). See `plot`.
`ylim`	a numeric (default value supplied). See `plot`.
`xlab`	a character (default value supplied). See `plot`.
`ylab`	a character (default value supplied). See `plot`.
`main`	a character (default value supplied). See `plot`.
`lwd`	line width used to plot the estimated density. See `plot`.
`col`	color used to plot the estimated density. See `plot`.
`...`	see `plot`.

Details

When confidence bands are requested they are obtained from the qunatiles of the `Poisson` distribution.

When a 2 elements vector is used as `breaks` argument it is interpreted as specifying a bin width (first element if elements are unnamed, `"bw"` element otherwise) and a step (second element if elements are unnamed, `"step"` element otherwise). The idea is then to obtain a smoother looking PSTH by counting spikes within overlapping bins. That is if the center of the ith bin is xi the one of the (i+1)th bin will be xi + step.

Value

When `plot` is set to `FALSE` in `psth`, a list of class `psth` is returned and no plot is generated. This list has the following components:

`freq`	a vector containing the instantaneous firing rate.
`ciUp`	a vector with the upper limit of the confidence band.
`ciLow`	a vector with the lower limit of the confidence band.
`breaks`	a numeric vector with the breaks in between which spikes were counted. Similar to the component of the same name returned by `hist`.
`mids`	a numeric vector with the mid points of `breaks`. Similar to the component of the same name returned by `hist`.
`counts`	a matrix with as many rows as components in `repeatedTrain` and as many columns as bins. Each element of the matrix contains the number of spikes falling in a given trial in a given bin.
`nbTrials`	the number of stimulations.
`call`	the matched call.

When `plot` is set to `TRUE` nothing is returned and a plot is generated as a side effect. Of course the same occurs upon calling `plot.psth` with a `psth` object argument.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Gerstein, George L. and Kiang, Nelson Y.-S. (1960) An Approach to the Quantitative Analysis of Electrophysiological Data from Single Neurons. *Biophysical Journal* **1**: 15–28. `http://www.pubmedcentral.nih.gov/articlerender.fcgi?tool=pubmed\&pubmedid=13704760`

Kalbfleisch, J. G. (1985) *Probability and Statistical Inference. Volume 2: Statistical Inference*. Springer-Verlag.

See Also

`as.repeatedTrain`, `is.repeatedTrain`, `print.repeatedTrain`, `summary.repeatedTrain`, `raster`

Examples

```
## Load Vanillin responses data (first cockroach data set)
data(CAL1V)
## convert them into repeatedTrain objects
## The stimulus command is on between 4.49 s and 4.99s
CAL1V <- lapply(CAL1V,as.repeatedTrain)
## look at the individual raster plots
plot(CAL1V[["neuron 1"]],stimTimeCourse=c(4.49,4.99),main="N1")
## Create a simple black and white PSTH for neuron 1
psth(CAL1V[["neuron 1"]],stimTimeCourse=c(4.49,4.99),breaks=20)
## Rebuilt the same PSTH but with red confidence bands
psth(CAL1V[["neuron 1"]],stimTimeCourse=c(4.49,4.99),breaks=20,colCI=2)
## Make the PSTH smoother
psth(CAL1V[["neuron 1"]],stimTimeCourse=c(4.49,4.99),breaks=c(bw=0.5,step=0.05),colCI=2)
## Make a plot with PSTHs from 4 neurons superposed
## First get lists containing PSTHs from each neuron
psth1 <- psth(CAL1V[["neuron 1"]],breaks=c(bw=0.5,step=0.05),plot=FALSE)
psth2 <- psth(CAL1V[["neuron 2"]],breaks=c(bw=1,step=0.1),plot=FALSE)
psth3 <- psth(CAL1V[["neuron 3"]],breaks=c(bw=0.5,step=0.05),plot=FALSE)
psth4 <- psth(CAL1V[["neuron 4"]],breaks=c(bw=2,step=0.2),plot=FALSE)
## Get the maximal frequency to display
maxFreq <- max(max(psth1$ciUp),max(psth2$ciUp),max(psth3$ciUp),max(psth4$ciUp))
## Build plot
plot(c(0,10),c(0,75),type="n",
     xaxs="i",yaxs="i",xlab="Time (s)",
     ylab="Freq. (Hz)",
     main="PSTHs from 4 simultaneously recorded neurons",
     sub="20 stimulations with vanillin were used.")
## Add rectangle corresponding to stimulation command
rect(4.49,0,4.99,75,col="grey80",lty=0)
## Add the neurons PSTHs as confidence bands
polygon(c(psth1$mids,rev(psth1$mids)),c(psth1$ciLow,rev(psth1$ciUp)),col=1,border=NA)
polygon(c(psth2$mids,rev(psth2$mids)),c(psth2$ciLow,rev(psth2$ciUp)),col=2,border=NA)
polygon(c(psth3$mids,rev(psth3$mids)),c(psth3$ciLow,rev(psth3$ciUp)),col=3,border=NA)
polygon(c(psth4$mids,rev(psth4$mids)),c(psth4$ciLow,rev(psth4$ciUp)),col=4,border=NA)
legend(0.1,maxFreq,legend=paste("neuron",1:4),lty=1,col=1:4,bty="n")
```

`purkinjeCellData` *Spike Trains of a Purkinje Cells (PC) Recorded in Control Conditions and With Bath Applied Bicuculline*

Description

An object of class `"SpikeTrain"`. Spontaneous discharge of a single PC recorded during 300 s in normal saline conditions and during 300 s in the presence of 25 μM bath applied bicuculline.

Usage

```
data(sPK)
data(mPK)
```

Format

sPK is a named list with 2 components ("ctl", "bicu". Each component contains the spike train (ie, action potentials occurrence times) of one Purkinje cell recorded during 300 s of spontaneous activity in control ("ctl") condition and with bath applied bicuculline ("bicu"). *Times are expressed in seconds.*

mPK is a named list with 8 components ("neuron 1", "neuron 2", ..., "neuron 8". Each component is itself a list with the spike train (ie, action potentials occurrence times) of one Purkinje cell recorded during 300 s of spontaneous activity in control ("ctl") condition and with bath applied bicuculline ("bicu"). *Times are expressed in seconds.*

Details

The recording contained in sPK was done in cell-attached mode. The one in mPK was done with a NeuroNexus silicon probe.

Bicuculline is a GABAA receptor antagonist. It blocks all GABAA inhibition.

Source

Recording and spike sorting performed by Matthieu Delescluse at the Cerebral Physiology Lab, CNRS UMR 8118: http://www.biomedicale.univ-paris5.fr/physcerv/physiologie_cerebrale.htm.

Examples

```
## Not run:
## load spontaneous data of 1 Purkinje cell
## recorded in cell attached mode from a cerebellar
## slice in control and bath applied bicuculline conditions
data(sPK)
## coerce data to spikeTrain objects
sPK <- lapply(sPK,as.spikeTrain)
## Get a summary of the ctl data
summary(sPK[["ctl"]])
## Look at the control train
## Don't show the rug plot for clarity
plot(sPK[["ctl"]],addRug=FALSE)
## Generate the renewal test plot taking into account
## the size of the data set (a lot of spikes!).
renewalTestPlot(sPK[["ctl"]],d=10,orderPlotPch=".",lag.max=250)
## Get a summary of the bicu data
summary(sPK[["bicu"]])
## Look at the control train
## Don't show the rug plot for clarity
plot(sPK[["bicu"]],addRug=FALSE)
## Generate the renewal test plot taking into account
## the size of the data set (a lot of spikes!).
renewalTestPlot(sPK[["bicu"]],d=10,orderPlotPch=".",lag.max=250);par(oldpar)
## This time the data are NOT stationary. This is seen clearly on a acf
## plot with very large lag.max
acf.spikeTrain(sPK[["bicu"]],lag.max=2000)

## End(Not run)
```

`qqDuration` *Quantile-Quantile Plot For Fitted Duration Distributions*

Description

Produces a QQ plot of empirical against theoretical quantiles of one of the following duration distributions: inverse Gaussian, log normal, log logistic, refractory exponential, gamma, weibull.

Usage

```
qqDuration(durationFit, CI = c(0.95, 0.99),
           type = "l", xlab, ylab, main, sub,
           ylim, dataLwd = 2, ablineCol = 2, ...)
```

Arguments

`durationFit`	a `durationFit` object, that is, a list returned by one of these functions: `invgaussMLE`, `lnormMLE`, `llogisMLE`, `rexpMLE`, `gammaMLE`, `weibullMLE`.
`CI`	a numeric vector with at most tow components, the confidence intervals to be drawn. If `NULL`, intervals are not drawn.
`type, xlab, ylab, main, sub, ylim`	see `plot`, default values are provided if arguments are missing.
`dataLwd`	non negative integer, the width of the line used to draw the data.
`ablineCol`	color of the diagonal.
`...`	additional arguments passed to `plot`.

Details

If the data to which the model was fitted have censored events, the latter are not used to build the empirical quantiles.

Value

Nothing is returned, the function is used for its side effect, a plot is generated.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`compModels`, `invgaussMLE`, `lnormMLE`, `llogisMLE`, `rexpMLE`, `gammaMLE`, `weibullMLE`

Examples

```
## Not run:
## Simulate a sample with 100 events from an inverse Gaussian
set.seed(1102006,"Mersenne-Twister")
mu.true <- 0.075
sigma2.true <- 3
sampleSize <- 100
```

```
sampIG <- rinvgauss(sampleSize,mu=mu.true,sigma2=sigma2.true)
## Fit it with an inverse Gaussian Model
sampIGmleIG <- invgaussMLE(sampIG)
## draw the QQ plot on a log scale
qqDuration(sampIGmleIG,log="xy")
## Fit it with a log normal Model
sampIGmleLN <- lnormMLE(sampIG)
## draw the QQ plot on a log scale
qqDuration(sampIGmleLN,log="xy")
## Fit it with a gamma Model
sampIGmleGA <- gammaMLE(sampIG)
## draw the QQ plot on a log scale
qqDuration(sampIGmleGA,log="xy")
## Fit it with a Weibull Model
sampIGmleWB <- weibullMLE(sampIG)
## draw the QQ plot on a log scale
qqDuration(sampIGmleWB,log="xy")
## Fit it with a refractory exponential Model
sampIGmleRE <- rexpMLE(sampIG)
## draw the QQ plot on a log scale
qqDuration(sampIGmleRE,log="xy")
## Fit it with a log logisitc Model
sampIGmleLL <- llogisMLE(sampIG)
## draw the QQ plot on a log scale
qqDuration(sampIGmleLL,log="xy")
## End(Not run)
```

raster *Generate a Raster Plot*

Description

Given a list of spike trains (or a `repeatedTrain` object) where each train was acquired during, say, one presentation of a given stimulus, a raster plot is generated. If stimulus time properties are specified, the stimulus application time also appears on the plot.

Usage

```
## S3 method for class 'repeatedTrain':
plot(x, stimTimeCourse = NULL,
       colStim = "grey80", xlim, pch, xlab, ylab, main, ...)
raster(x, stimTimeCourse = NULL, colStim = "grey80",
       xlim, pch, xlab, ylab, main, ...)
```

Arguments

`x`	a `repeatedTrain` object or a list which can be coerced to such an object.
`stimTimeCourse`	`NULL` (default) or a two elements vector specifying the time boundaries (in s) of a stimulus presentation.
`colStim`	the background color used for the stimulus.
`xlim`	a numeric (default value supplied). See `plot`.

`pch`	data symbol used for the spikes. See `plot`.
`xlab`	a character (default value supplied). See `plot`.
`ylab`	a character (default value supplied). See `plot`.
`main`	a character (default value supplied). See `plot`.
`...`	see `plot`.

Details

Basic raster plot stuff.

Value

Nothing is returned `raster` is used for its side effect, a plot is generated on the current graphical device.

Note

Brillinger (1992) calls these plots "rastor" instead of raster...

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Brillinger, David R. (1992) Nerve Cell Spike Train Data Analysis: A Progression of Technique. *JASA* **87**: 260–271.

See Also

`as.repeatedTrain`, `is.repeatedTrain`, `print.repeatedTrain`, `summary.repeatedTrain`, `psth`

Examples

```
## Load Vanillin responses data (first cockroach data set)
data(CAL1V)
## convert them into repeatedTrain objects
## The stimulus command is on between 4.49 s and 4.99s
CAL1V <- lapply(CAL1V,as.repeatedTrain)
## look at the individual raster plots
raster(CAL1V[["neuron 1"]],stimTimeCourse=c(4.49,4.99),main="N1")
plot(CAL1V[["neuron 2"]],stimTimeCourse=c(4.49,4.99),main="N2")
plot(CAL1V[["neuron 3"]],stimTimeCourse=c(4.49,4.99),main="N3")
plot(CAL1V[["neuron 4"]],stimTimeCourse=c(4.49,4.99),main="N4")
```

`rateEvolution` *Evaluates and Plots a Spike Train Firing Rate's Evolution*

Description

`rateEvolution` evaluates and `plot.rateEvolution` plots the firing rate evolution of a `spikeTrain` object. The evaluation is done by convolving the spike train with a kernel like in `density` estimation.

Usage

```
rateEvolution(x, bw, kernel = c("gaussian", "epanechnikov",
                                "rectangular", "triangular",
                                "biweight", "cosine", "optcosine"),
              n = 512, from, to, na.rm = FALSE, ...)
## S3 method for class 'rateEvolution':
plot(x, main = NULL, xlab = NULL, ylab = "Rate (Hz)",
                   type = "l", zero.line = TRUE, ...)
```

Arguments

`x`	a `spikeTrain` object or an object which can be coerced to it for `rateEvolution` or a `rateEvolution` object for `plot.rateEvolution`.
`bw`	the kernel bin width in seconds. If `missing` it is set to 10 times the median inter-spike interval of `x`.
`kernel`	see `density`.
`n`	see `density`.
`from`	see `density`.
`to`	see `density`.
`na.rm`	see `density`.
`main`	see `plot.density`.
`xlab`	see `plot.density`.
`ylab`	see `plot.density`.
`type`	see `plot.density`.
`zero.line`	see `plot.density`.
`...`	see `density` and `plot.density`.

Details

`rateEvolution` is mainly a wrapper for `density` which also adjusts the result of the latter such that the y component of the returned list is an instantaneous firing rate. If the length of `x` is smaller or equal to 1 and if `from` or `to` is (are) `missing` the returned object has then each of its components set to `NA` except `data.name` (see below). If the length of `x` is smaller or equal to 1 and if both `from` and `to` are specified a `missing` `bw` is then set to 3 times the spacing between the points of the regular grid on which the density is evaluated.

`plot.rateEvolution` is also a wrapper for `plot.density` which only adjust the default value of some arguments.

Value

`rateEvolution` returns a LIST of class `rateEvolution` which inherits from class `density`.

`x`	the `n` coordinates of the points where the density is estimated. See `density`.
`y`	the estimated rate (in 1/s). These will be non-negative, but can be zero.
`bw`	the bandwidth used.
`n`	the sample size after elimination of missing values.
`call`	the call which produced the result.
`data.name`	the deparsed name of the `x` argument.
`has.na`	logical, for compatibility (always `FALSE`).

`plot.rateEvolution` is called for its side effect: a plot is generated.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`as.spikeTrain`, `density`, `plot.density`, `mkREdf`

Examples

```
## load Purkinje cell data recorded in cell-attached mode
data(sPK)
## coerce sPK to a spikeTrain object
sPK <- lapply(sPK, as.spikeTrain)
## get the rate evolution in ctl condition
sPKreCTL <- rateEvolution(sPK[["ctl"]])
## plot the result
plot(sPKreCTL)
## check the bin width which was actually used
sPKreCTL$bw
## look at the effect of a 10 times larger bw
plot(rateEvolution(sPK[["ctl"]],bw=10*sPKreCTL$bw))
## look at the effect of a 10 times smaller one
plot(rateEvolution(sPK[["ctl"]],bw=sPKreCTL$bw/10))
## get the rate evolution in bicuculline conditions
sPKreBICU <- rateEvolution(sPK[["bicu"]])
## plot results
plot(sPKreBICU,col=2)
## add the ctl rate evolution
lines(sPKreCTL)
```

`renewalTestPlot` *Non-Parametric Tests for Renewal Processes*

Description

Performs and displays rank based tests checking if a spike train is a renewal process

Usage

```
renewalTestPlot(spikeTrain, lag.max = NULL,
                d=max(c(2,sqrt(length(spikeTrain)) %/% 5)),
                orderPlotPch=ifelse(length(spikeTrain)<=600,1,"."),
                ...)
```

Arguments

`spikeTrain`	a `spikeTrain` object or a vector which can be coerced to such an object.
`lag.max`	argument passed to `acf.spikeTrain`.
`d`	an integer >= 2, the number of divisions used for the Chi 2 test. The default value is such that under the null hypothesis at least 25 events should fall in each division.
`orderPlotPch`	`pch` argument for the order plots.
`...`	additional arguments passed to function `chisq.test`.

Details

`renewalTestPlot` generates a 4 panel plot. The 2 graphs making the top row are qualitative and display the rank of inter-spike interval (ISI) k+1 versus the rank of ISI k (left graph) and the rank of ISI k+2 versus the one of ISI k (right graph). The bottom left graph displays the autocorrelation function of the ISIs and is generated by a call to `acf.spikeTrain`. The bottom right graph display the result of a Chi square test performed on the ranks at different lags. More precisely, for each considered lag j (from 1 to `lag.max`) the square within which the rank of ISI k+1 vs the one of ISI k is found is splited in d^2 cells. This decomposition into cells is shown on the two graphs of the top row. Under the renewal process hypothesis the points should be uniformly distributed with a density $\frac{N}{d^2}$, where N is the number of ISIs. The sum other rows and other columns is moreover exactly $\frac{N}{d}$. The upper graphs are therefore graphical displays of two-dimensional contingency tables. A chi square test for two-dimensional contingency tables (function `chisq.test`) is performed on the table generated at each lag j. The resulting Chi 2 value is displayed vs the lag. The 95% confidence region appears as a clear grey rectangle, the value falling within this region appear as black dots and the ones falling out appear as dark grey triangles.

Value

Nothing is returned, the function is used for its side effect: a plot is generated.

Note

You should not use a too large value for `d` otherwise the Chi 2 values will be too approximative and warnings will be printed. If your process is a renewal process you should have on average 5% of the points on the bottom right graph appearing as dark triangles.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`acf`, `varianceTime`, `acf.spikeTrain`

Examples

```
## Apply the test of Ogata (1988) shallow shock data
data(ShallowShocks)
renewalTestPlot(ShallowShocks$Date,d=3)

## Apply the test to the second and third neurons of the cockroachAlSpont
## data set
## load spontaneous data of 4 putative projection neurons
## simultaneously recorded from the cockroach (Periplaneta
## americana) antennal lobe
data(CAL1S)
## convert data into spikeTrain objects
CAL1S <- lapply(CAL1S,as.spikeTrain)
## look at the individual trains
## first the "raw" data
CAL1S[["neuron 1"]]
## next some summary information
summary(CAL1S[["neuron 1"]])
## next the renewal tests
renewalTestPlot(CAL1S[["neuron 1"]])

## Simulate a renewal log normal train with 500 isi
isi.nb <- 500
train1 <- c(cumsum(rlnorm(isi.nb+1,log(0.01),0.25)))
## make the test
renewalTestPlot(train1)

## Simulate a (non renewal) 2 states train
myTransition <- matrix(c(0.9,0.1,0.1,0.9),2,2,byrow=TRUE)
states2 <- numeric(isi.nb+1) + 1
for (i in 1:isi.nb) states2[i+1] <- rbinom(1,1,prob=1-myTransition[states2[i],])+1
myLnormPara2 <- matrix(c(log(0.01),0.25,log(0.05),0.25),2,2,byrow=TRUE)
train2 <-
cumsum(rlnorm(isi.nb+1,myLnormPara2[states2,1],myLnormPara2[states2,2]))
## make the test
renewalTestPlot(train2)
```

`reportHTML.gam` *Generates a Report in HTML Format from a STAR gam Object*

Description

Writes the result of a `gam` fit in an html file.

Usage

```
## S3 method for class 'gam':
reportHTML(object, filename, extension = "html",
           directory = getwd(), Title,
           neuron, neuronEvts, ...)
```

Arguments

`object`	an object returned by `gam`.
`filename`	a character string. The generic name of all the files (html, png as well as `R` data files which will be generated. See also `HTMLInitFile`.
`extension`	see `HTMLInitFile`.
`directory`	the full or relative path to the directory where the results are going to be stored. See also `HTMLInitFile`.
`Title`	See `HTMLInitFile`. If missing a default value baed on `filename` is provided.
`neuron`	a character string describing to which the analysis refers and used for the titles of the interaction plots (see `plot.frt`).
`neuronEvts`	a named list with the `event` variable from the data frame returned by `mkGLMdf` and corresponding to the other neurons recorded simultaneously. One list element per neuron.
`...`	Not used, only there for compatibilty with the generic method definition.

Details

A summary (`summary.gam`) of `object` is added to the report. A plot of the spike train after time transformation `transformedTrain` comes next followed by a renewal test plot (`renewalTestPlot`) of the spike train on the time transformed scale. The "usual" Ogata's tests plots (`plot.transformedTrain`) are added. Then if other trains are provided as a named list via argument `neuronEvts`, interactions plots (`plot.frt`) are built showing both the survivor function and the Berman's test. The report ends with the `call` which generated `object`.

Value

Nothing is returned, an html file and figures in png format are written to disk.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`mkGLMdf`, `gam`, `gam.check`, `frt`, `transformedTrain`, `plot.transformedTrain`, `summary.transform`

Examples

```
## Not run:
## load e070528spont data set
data(e070528spont)
## make a data frame for gam using a 2 ms bin width
spontDF <- mkGLMdf(e070528spont,0.002,0,60)
## make data frames specific of each neuron
```

```
n1.spontDF <- spontDF[spontDF$neuron=="1",]
n2.spontDF <- spontDF[spontDF$neuron=="2",]
n3.spontDF <- spontDF[spontDF$neuron=="3",]
n4.spontDF <- spontDF[spontDF$neuron=="4",]
## save space by removing the now redundant spontDF
rm(spontDF)
## fit neuron 1 using the gam representation of a
## renewal process and a binomial model
n1.spontFit1 <- gam(event ~ s(lN.1,k=25,bs="cr"),data=n1.spontDF,family=binomial())
## create a list with the discretized spike times of the 3 other neurons
preN1 <- list(n2=with(n2.spontDF,event),n3=with(n3.spontDF,event),n4=with(n4.spontDF,ever
## generate the report
reportHTML(n1.spontFit1,"e070528spontN1gFit",neuron="1",neuronEvts=preN1)
## End(Not run)
```

`reportHTML` *Generic Function for Automatic HTML Report Generation*

Description

When a standard analysis is applied to some object it is useful to keep all the plots and summaries related to that analysis in a single place where they can be easily accessed and visualized. An html file containing the report of this analysis is ideally suited for that. The methods `reportHTML` generate such reports.

Usage

```
reportHTML(object, filename, extension, directory, Title, ...)
```

Arguments

`object`	an object from which the report is going to be generated, perhaps following some standard analysis procedure.
`filename`	a character string. The generic name of all the files (html, png as well as `R` data files which will be generated. See also `HTMLInitFile`.
`extension`	see `HTMLInitFile`.
`directory`	the full or relative path to the directory where the results are going to be stored. See also `HTMLInitFile`.
`Title`	See `HTMLInitFile`. If missing a default value baed on `filename` is provided.
`...`	additional parameters passed to the functions internally called by the actual methods.

Value

Nothing is returned, an html file and figures in png format are written to disk together with the `R` variables generated during the analysis , if an analysis was performed.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

See Also

`reportHTML.spikeTrain`, `reportHTML.repeatedTrain`, `reportHTML.gam`

Examples

```
##
```

`reportHTML.repeatedTrain` *Performs Basic Spike Train Analysis and Generates a Report in HTML Format from a repeatedTrain Object*

Description

Performs a "standard" analysis on a `repatedTrain` object, writes results to disk and generates a report in html format.

Usage

```
## S3 method for class 'repeatedTrain':
reportHTML(object, filename, extension = "html",
           directory = getwd(), Title, binSize = 0.025,
           k = 100, bs = "tp", stimTimeCourse = NULL,
           colCI = 2, doGamCheck = TRUE,
           doTimeTransformation = TRUE, ...)
```

Arguments

`object`	a `repeatedTrain` object.
`filename`	a character string. The generic name of all the files (html, png as well as `R` data files which will be generated. See also `HTMLInitFile`.
`extension`	see `HTMLInitFile`.
`directory`	the full or relative path to the directory where the results are going to be stored. See also `HTMLInitFile`.
`Title`	See `HTMLInitFile`. If missing a default value baed on `filename` is provided.
`binSize, k, bs`	See `spsth`.
`stimTimeCourse`	See `plot.repeatedTrain` and `plot.spsth`.
`colCI`	See `plot.spsth`.
`doGamCheck`	Should function `gam.check` be used on the inhomogenous Poisson fit performed to obtain the smooth PSTH?
`doTimeTransformation`	Should the estimated integrated intensity be used to perform a time transformation and generate Ogata's test plots?
`...`	Not used, only there for compatibilty with the generic method definition.

Details

A raster plot is added first to te report (`plot.transformedTrain`) with a smooth PSTH (`spsth`) superposed. The summary of the inhomogenous Poisson fit leading the the smooth PSTH is added next together with a short summary describing how accurate the hypothesis of constant intensity/rate made during the pre-processing of the `repeatedTrain` was in view of the estimated rate. Check `spsth` for details. A plot of the smooth PSTH with approximate 95% CI is added. If `doGamCheck` is set to `TRUE` a diagnostic plot for the fitted inhomogenous Poisson model is added. If `doTimeTransformation` is set to `TRUE` the estimated integrated intensity is used to perform a time transformation and Ogata's test plots are generated.

A `R` data file (`filename.rda`) is also generated with the following objects:

- `PoissonF`: the `gamObject` containing the result of the `gam` fit with the inhomogenous Poisson model.
- `Lambda`: the integrated intensity of `repeatedTrain` under the inhomogenous Poisson model hypothesis. If `doTimeTransformation` was set to `TRUE`.
- `fct`: the matched call.

Value

Nothing is returned, an html file and figures in png format are written to disk together with the `R` variables generated during the analysis.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`as.repeatedTrain`, `plot.repeatedTrain`, `summary.repeatedTrain`, `spsth`, `transformedTrain`, `plot.transformedTrain`, `summary.transformedTrain`, `gam`, `gam.check`, `frt`

Examples

```
## load e070528citronellal data set
data(e070528citronellal)
## make a standard analysis on the first neuron
reportHTML(e070528citronellal[["neuron 1"]],"e070528citronellalN1",stim=c(6.14,6.64))
```

`reportHTML.spikeTrain` *Performs Basic Spike Train Analysis and Generates a Report in HTML Format from a spikeTrain Object*

Description

Performs a "standard" analysis on a `spikeTrain` object, computing some cross-correlation statistics if additional `spikeTrain` objects are provided, writes results to disk and generates a report in html format.

Usage

```
## S3 method for class 'spikeTrain':
reportHTML(object, filename, extension = "html",
           directory = getwd(), Title, forceTT = TRUE,
           digits = 3, timeUnit = "s", otherST,
           laglim = c(-0.1, 0.1),
           cch = c("both", "scch", "cch"),
           doGamCheck = TRUE, k = 100, bs = "tp",
           nbEvtPerBin = 50, ...)
```

Arguments

`object`	a `spikeTrain` object.
`filename`	a character string. The generic name of all the files (html, png as well as `R` data files which will be generated. See also `HTMLInitFile`.
`extension`	see `HTMLInitFile`.
`directory`	the full or relative path to the directory where the results are going to be stored. See also `HTMLInitFile`.
`Title`	See `HTMLInitFile`. If missing a default value baed on `filename` is provided.
`forceTT`	Should a time transformation be performed and the `compModels` plots be generated even if none of the six renewal models fits the data?
`timeUnit, digits`	see `summary.spikeTrain`.
`otherST`	a named list of `spikeTrain` objects from simultaneously recorded neurons or nothing.
`laglim`	see `lockedTrain`.
`cch`	if `otherST` is given (ie, not missing) cross-intensity plots will be made using the neuron of `spikeTrain` as a reference. Should smooth version of the cross-intensity be computed (`"scch"`), a "classical" one (`"cch"`) or both (`"both"`). Only the first element of `cch` is used.
`doGamCheck`	if smooth estimates are requested, should function `gam.check` be used on them?
`k`	see `slockedTrain`.
`bs`	`slockedTrain`.
`nbEvtPerBin`	a number of event per bin used in a way similar to the argument with the same name in `jpsth` when a bining is used for pre-processing.
`...`	Not used, only there for compatibilty with the generic method definition.

Details

A spike train plot (`plot.spikeTrain`) is performed first. The summary (`summary.spikeTrain`) is computed next and part of its output is written to the html file. The renewal tests are then carried out and their results added (`renewalTestPlot`). The six duration distributions are fitted (`compModels` with argument `plot` set to `FALSE`) and the best one is used to apply a time transformation to `spikeTrain`. The Ogata's tests are applied (`summary.transformedTrain`) and if they are all within the 99% confidence interval, the result of the transformation is plotted (`plot.transformedTrain`) as well as all the Q-Q plots of `compModels`. If `forceTT` is set to `TRUE` (default), then these last two plots are added even if the best model does not pass the tests.

If other spikeTrain objects are provided as a named list via argument otherST, then cross-correlation/cross-intensity functions are estimated; Two estimations methods are available, the classical histogram and a smooth version of it. Argument cch controls if a single estimation is performed or if both are performed. If the smooth version is requested a summary of the gam fit is printed. Moreover if argument doGamCheck is set to TRUE then check plots (gam.check) are added to the report.

A R data file (filename.rda) is also generated with the following objects:

- cm: the result of compModels.
- bestFit: the durationDistribution object returned obtained by fitting the best model among the 6.
- Lambda: the integrated intensity of spikeTrain with the best model.
- fct: the matched call.
- cchL: if other trains were provided and if argument cch was set to "both" or to "cch". A list with as many components as the otherST argument. Each component is the a hist.lockedTrain object.
- scchL: if other trains were provided and if argument cch was set to "both" or to "scch". A list with as many components as the otherST argument. Each component is the a slockedTrain object.

Value

Nothing is returned, an html file and figures in png format are written to disk together with the R variables generated during the analysis.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

as.spikeTrain, plot.spikeTrain, summary.spikeTrain, renewalTestPlot, plot.spikeTrain, compModels, transformedTrain, plot.transformedTrain, summary.transformedTrain, gam, gam.check, lockedTrain, slockedTrain

Examples

```
## load e070528spont data set
data(e070528spont)
## perform a standard analysis on neuron 1, looking for cross-correlations
## with the 3 other neurons up to lag +/- 250 ms.
## Store the results under the generic name: e070528spontN1
reportHTML(e070528spont[["neuron 1"]],"e070528spontN1",otherST=e070528spont[-1],laglim=c(
## Neuron 1 of e070528spont is exceptional in that it can be well
## described by a renewal process...
```

`rexpMLE`	*Maximum Likelihood Parameter Estimation of a Refractory Exponential Model with Possibly Censored Data*

Description

Estimate refractory exponential model parameters by the maximum likelihood method using possibly censored data.

Usage

```
rexpMLE(yi, ni = numeric(length(yi)) + 1,
        si = numeric(length(yi)) + 1)
```

Arguments

`yi`	vector of (possibly binned) observations or a `spikeTrain` object.
`ni`	vector of counts for each value of `yi`; default: `numeric(length(yi))+1`.
`si`	vector of counts of *uncensored* observations for each value of `yi`; default: `numeric(length(yi))+1`.

Details

The MLE are available in closed form even in the censored case for this model. The likelihood function cannot be differentiated with respect to the `rp` (refractory period) parameter at the maximum. COnfidence intervals for this parameter are therefore not available.

Value

A list of class `durationFit` with the following components:

`estimate`	the estimated parameters, a named vector.
`se`	the standard errors, a named vector.
`logLik`	the log likelihood at maximum.
`r`	a function returning the log of the relative likelihood function.
`mll`	a function returning the opposite of the log likelihood function using the log of the parameters.
`call`	the matched call.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

See Also

`drexp`, `invgaussMLE`, `lnormMLE`, `gammaMLE`, `weibullMLE`

Examples

```
## Not run:
## Simulate sample of size 100 from a refractory exponential distribution
set.seed(1102006,"Mersenne-Twister")
sampleSize <- 100
rate.true <- 20
rp.true <- 0.01
sampRE <- rrexp(sampleSize,rate=rate.true,rp=rp.true)
sampREmleRE <- rexpMLE(sampRE)
rbind(est = sampREmleRE$estimate,se = sampREmleRE$se,true = c(rate.true,rp.true))

## make a parametric boostrap to check the distribution of the deviance
nbReplicate <- 10000
system.time(
            devianceRE100 <- replicate(nbReplicate,{
              sampRE <- rrexp(sampleSize,rate=rate.true,rp=rp.true)
              sampREmleRE <- rexpMLE(sampRE)
              -2*sampREmleRE$r(rate.true,rp.true)
            }
                                          )
            )[3]

## Get 95 and 99% confidence intervals for the QQ plot
ci <- sapply(1:nbReplicate,
                 function(idx) qchisq(qbeta(c(0.005,0.025,0.975,0.995),
                                             idx,
                                             nbReplicate-idx+1),
                                        df=2)
             )
## make QQ plot
X <- qchisq(ppoints(nbReplicate),df=2)
Y <- sort(devianceRE100)
X11()
plot(X,Y,type="n",
     xlab=expression(paste(chi[2]^2," quantiles")),
     ylab="MC quantiles",
     main="Deviance with true parameters after ML fit of refractory Poisson data",
     sub=paste("sample size:", sampleSize,"MC replicates:", nbReplicate)
     )
abline(a=0,b=1)
lines(X,ci[1,],lty=2)
lines(X,ci[2,],lty=2)
lines(X,ci[3,],lty=2)
lines(X,ci[4,],lty=2)
lines(X,Y,col=2)
## End(Not run)
```

`ShallowShocks` *Shallow Shocks (M >= 6.0) in OFF Tohoku Area for 1885-1980*

Description

Earthquakes data used by Yosihiko Ogata in his 1988 JASA paper.

Usage

```
data(ShallowShocks)
```

Format

A `data.frame` with the following variables:

year:	year of occurrence.
month:	month of occurrence.
day:	day of occurrence.
hour:	hour of occurrence.
minute:	minute of occurrence.
magnitude:	magnitude on Richter's scale.
type:	type of earthquake: `main` (shock), `foreshock`, `aftershock`; according to Utsu.
Date:	date in days starting from January 1st 1885.
energy.sqrt:	square root of the energy expressed in erg.

Details

Quakes 213 and 214 were given exactly the same dates in Ogata (1988). Quake 214 has here been delayed by 1 minute.

Source

Ogata (1988) Table 1, pp 14-15.

References

Ogata, Yosihiko (1988) Statistical Models for Earthquake Occurrences and Residual Analysis for Point Processes. *Journal of the American Statistical Association* **83**: 9-27.

Examples

```
data(ShallowShocks)
## Reproduce Fig. 2 of Ogata 1988
layout(matrix(1:3, nrow = 3))
plot(ShallowShocks$Date,
     cumsum(ShallowShocks$energy.sqrt) / 10^13,
     type ="l",
     xlab = "",
     ylab = "",
     main = "Cumulative square root of energy")
plot(ShallowShocks$Date,
     cumsum(1+numeric(dim(ShallowShocks)[1])),
     type ="l",
     xlab = "",
     ylab = "",
     main = "Cumulative number of shocks")
plot(ShallowShocks$Date,
     ShallowShocks$magnitude,
     type = "h",
     ylim = c(5,9),
     xlab = "Time (days)",
     ylab = "",
     main = "Magnitude vs Occurrence time")
```

`slockedTrain` *Function to Smooth a lockedTrain Object and Related Methods*

Description

Smooths a `lockedTrain` object using a `gam` model with the Poisson family after binning the object.

Usage

```
slockedTrain(lockedTrain, bw = 0.001, bs = "cr", k = 100, ...)
## S3 method for class 'slockedTrain':
print(x, ...)
## S3 method for class 'slockedTrain':
summary(object, ...)
## S3 method for class 'slockedTrain':
plot(x, xlab, ylab, main, xlim, ylim, col, lwd, ...)
```

Arguments

`lockedTrain`	a `lockedTrain` object.
`bw`	the bin width (in s) used to generate the observations on which the gam fit will be performed. See details below.
`bs`	the type of splines used. See `s`.
`k`	the dimension of the basis used to represent the smooth psth. See `s`.
`x`	an `slockedTrain` object.
`object`	an `slockedTrain` object.
`xlim`	a numeric (default value supplied). See `plot`.
`ylim`	a numeric (default value supplied). See `plot`.
`xlab`	a character (default value supplied). See `plot`.
`ylab`	a character (default value supplied). See `plot`.
`main`	a character (default value supplied). See `plot`.
`lwd`	line width used to plot the estimated density. See `plot`.
`col`	color used to plot the estimated density. See `plot`.
`...`	additional arguments passed to `gam` in `slockedTrain`. Not used in `print.slockedTrain` and `summary.slockedTrain`. Passed to `plot` in `plot.slockedTrain`.

Details

`slockedTrain` essentially generates a smooth version of the histogram obtained by `hist.lockedTrain`. The Idea is to build the histogram first with a "too" small bin width before fitting a regression spline to it with a Poisson distribution of the observed counts.

Value

A list of class `slockedTrain` is returned by `slockedTrain`. This list has the following components:

`gamFit`	the `gamObject` generated.
`Time`	the vector of bin centers.
`nRef`	the number of spikes in the reference train. See `hist.lockedTrain`.
`testFreq`	the mean frequency of the test neuron. See `hist.lockedTrain`.
`bwV`	the vector of bin widths used.
`CCH`	a logical which is `TRUE` if a cross-intensity was estimated and `FALSE` in the case of an auto-intensity.
`call`	the matched call.

`print.slockedTrain` returns the result of `print.gam` applied to the component `gamFit` of its argument.

`summary.slockedTrain` returns the result of `summary.gam` applied to the component `gamFit` of its argument.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Wood S.N. (2006) *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC Press.

See Also

`lockedTrain`, `plot.lockedTrain`, `gam`

Examples

```
## load e070528spont data set
data(e070528spont)
## create a lockedTrain object with neuron 1 as reference
## and neuron 3 as test up to lags of +/- 250 ms
lt1.3 <- lockedTrain(e070528spont[[1]],e070528spont[[3]],laglim=c(-1,1)*0.25)
## look at the cross raster plot
lt1.3
## build a histogram of it using a 10 ms bin width
hist(lt1.3,bw=0.01)
## do it the smooth way
slt1.3 <- slockedTrain(lt1.3)
plot(slt1.3)
## do some check on the gam fit
summary(slt1.3)
gam.check(gamObj(slt1.3))
```

`spsth` *Smooth Peri Stimulus Time Histogram Related Functions and Methods*

Description

Function `spsth` computes a smooth psth, while method `print.spsth` prints and `summary.spsth` summarises the `gamObject` contained in the returned `spsth` object and `plot.spsth` plots it.

Usage

```
spsth(repeatedTrain, binSize = 0.025, k = 100,
      bs = "tp", plot = TRUE,...)
## S3 method for class 'spsth':
print(x, ...)
## S3 method for class 'spsth':
summary(object, ...)
## S3 method for class 'spsth':
plot(x, stimTimeCourse = NULL, colStim = "grey80",
           colCI = NULL, xlab, ylab, main, xlim, ylim,
           lwd = 2, col = 1, ...)
```

Arguments

`repeatedTrain`	a `repeatedTrain` object or a list which can be coerced to such an object.
`binSize`	the bin size (in s) used to generate the observations on which the gam fit will be performed. See details below.
`k`	the dimension of the basis used to represent the smooth psth. See `s`.
`bs`	the type of splines used. See `s`.
`plot`	corresponding argument of `hist`. Should a plot be generated or not?
`object`	a `spsth` object.
`x`	a `spsth` object.
`stimTimeCourse`	`NULL` (default) or a two elements vector specifying the time boundaries (in s) of a stimulus presentation.
`colStim`	the background color used for the stimulus.
`colCI`	if not `NULL` (default) a confidence band is plotted with the specified color; two dashed lines are plotted otherwise.
`xlim`	a numeric (default value supplied). See `plot`.
`ylim`	a numeric (default value supplied). See `plot`.
`xlab`	a character (default value supplied). See `plot`.
`ylab`	a character (default value supplied). See `plot`.
`main`	a character (default value supplied). See `plot`.
`lwd`	line width used to plot the estimated density. See `plot`.
`col`	color used to plot the estimated density. See `plot`.
`...`	in `spsth`, if `plot` is set to `TRUE` then the ... are passed to `plot.spsth`. In `plot.spsth` they are passed to `plot` which is called internally. They are not used otherwise.

Details

For spsth, the raw data contained in repeatedTrain are pre-processed with hist using a bin size given by argument binSize. This binSize should be small "enough". That is, the rate of the aggregated train created by collapsing the spike times of the different trials onto a single "pseudo" spike train, should not change too much on the scale of binSize (see Ventura et al (2002) Sec. 4.2 p8 for more details).

Value

When plot is set to FALSE in spsth, a list of class spsth is returned and no plot is generated. This list has the following components:

freq	a vector containing the instantaneous firing rate in the middle of the "thin" bins used for preprocessing.
ciUp	a vector with the upper limit of a pointwise 95% confidence interval. Check predict.gam for details.
ciLow	a vector with the lower limit of a pointwise 95% confidence interval.
breaks	a vector with 2 elements the ealiest and the latest spike in repeatedTrain.
mids	a numeric vector with the mid points of the bins.
counts	a vector with the actual number of spikes in each bin.
nbTrials	the number of trials in repeatedTrain.
lambdaFct	a function of a single time argument returning the estimated intensity (or instantaneous rate) at its argument.
LambdaFct	a function of a single time argument returning the integrale of estimated intensity (or instantaneous rate) at its argument. That is, the integrated intensity. integrate is used by this function.
call	the matched call.

When plot is set to TRUE nothing is returned and a plot is generated as a side effect. Of course the same occurs upon calling plot.spsth with a spsth object argument.

print.spsth returns the result of print.gam applied to the gamObject generated by spsth and stored in the environment of both lambdaFct and LambdaFct.

summary.spsth returns the result of summary.gam applied to the gamObject generated by spsth and stored in the environment of both lambdaFct and LambdaFct.

Note

Most of the components of the list returned by spsth are not of direct interest for the user but they are used by, for instance, reportHTML.repeatedTrain.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Ventura, V., Carta, R., Kass, R. E., Gettner, S. N. and Olson, C. R. (2002) Statistical analysis of temporal evolution in single-neuron firing rates. *Biostatistics* **3**: 1–20.

Kass, R. E., Ventura, V. and Cai, C. (2003) Statistical smoothing of neuronal data. *Network: Computation in Neural Systems* **14**: 5–15.

Wood S.N. (2006) *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC Press.

See Also

psth, plot.psth, gam, print.gam, summary.gam, gam.check, reportHTML.repeatedTrain,

Examples

```
## Get the e070528citronellal data set into workspace
data(e070528citronellal)
## Compute spsth without a plot for neuron 1
## using a cubic regression spline
n1CitrSPSTH <- spsth(e070528citronellal[[1]],plot=FALSE,bs="cr")
## plot the result
plot(n1CitrSPSTH,stim=c(6.14,6.64),colCI=2)
## get a summary of the gam fit
summary(n1CitrSPSTH)
## perhaps get a more complete check wit gam.check
n1CitrSPSTHgo <- gamObj(n1CitrSPSTH)
gam.check(n1CitrSPSTHgo)
## It does not look too bad
## Now take a look at the observation on which the gam
## was actually performed
plot(n1CitrSPSTH$mids,n1CitrSPSTH$counts,type="l")
## put dots at the positions of the knots
X <- n1CitrSPSTHgo$smooth[[1]][["xp"]]
rug(X,col=2)
## Add the estimated smooth psth after proper scaling
theBS <- diff(n1CitrSPSTH[["mids"]])[1]
Y <- n1CitrSPSTH$lambdaFct(n1CitrSPSTH$mids)*theBS*n1CitrSPSTH$nbTrials
lines(n1CitrSPSTH$mids,Y,col=4,lwd=2)
```

`STAR-package` *Spike Train Analysis with R*

Description

Functions to analyze neuronal spike trains

Details

Package:	STAR
Type:	Package
Version:	0.1-5
Date:	2007-11-07
Depends:	survival, mgcv, R2HTML, sound
Suggests:	gam, lattice, ggplot2, HiddenMarkov
License:	GPL version 2 or newer
URL:	http://www.biomedicale.univ-paris5.fr/physcerv/C_Pouzat/STAR.html

Author(s)

Christophe Pouzat

Maintainer: Christophe Pouzat <christophe.pouzat@gmail.com>

`summary.transformedTrain` *Summary of transformedTrain Objects*

Description

Generates a concise summary of `transformedTrain` objects. It is mostly intended for use in batch processing situations where a decision to stop with the current model or go on with a more complicated one must be made automatically.

Usage

```
## S3 method for class 'transformedTrain':
summary(object, ...)
```

Arguments

`object`	a `transformedTrain` object.
`...`	additional arguments passed to `varianceTime`.

Details

`summary.transformedTrain` computes summary statistics corresponding to plot 1, 2 and 5 of `plot.transformedTrain`.

The first plot tests the uniformity of the spikes (transformed) times on the (transformed) observation window using a KS test. If the ecdf of the (transformed) times is within the 95% band then the first element of component `uniformOnTTime` of the returned list is set to `TRUE`. It is set to `FALSE` otherwise. The second component is relative to the 99% band.

The second plot tests the exponential distribution of the intervals between successive spikes transformed times. Again if the empirical curve stays within the 95, respectively 99%, confidence band, the first, respectively second, element of component `BermanTest` of the returned list is set to `TRUE`. It is set to `FALSE` otherwise.

The fifth plot tests that the variance is equal to the length of the (transformed) observation time for `object`, using point-wise CI. If n different observation times are defined over the whole observation window, we expect (1 - CI/100)*n points to be out with an approximate binomial distribution. For each CI defined (95 and 99%, by default), component `VarTime` of the returned list contains the probability of observing a number as large as or smaller than the one observed under the binomial null hypothesis.

Value

A `list` with the following 3 components:

`uniformOnTTime`	A two named components vector of boolean.
`BermanTest`	A two named components vector of boolean.

`VarTime` A named component vector with as many components as passed to `varianceTime` via the `...` argument with p-values of a binomial distribution.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Ogata, Yosihiko (1988) Statistical Models for Earthquake Occurrences and Residual Analysis for Point Processes. *Journal of the American Statistical Association* **83**: 9-27.

Brown, E. N., Barbieri, R., Ventura, V., Kass, R. E. and Frank, L. M. (2002) The time-rescaling theorem and its application to neural spike train data analysis. *Neural Computation* **14**: 325-346.

See Also

`transformedTrain`, `plot.transformedTrain`, `mkGLMdf`

Examples

```
## Not run:
## Let us consider neuron 1 of the CAL2S data set
data(CAL2S)
CAL2S <- lapply(CAL2S,as.spikeTrain)
CAL2S[["neuron 1"]]
renewalTestPlot(CAL2S[["neuron 1"]])
summary(CAL2S[["neuron 1"]])
## Make a data frame with a 4 ms time resolution
cal2Sdf <- mkGLMdf(CAL2S,0.004,0,60)
## keep the part relative to neuron 1, 2 and 3 separately
n1.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="1",]
n2.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="2",]
n3.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="3",]
## remove unnecessary data
rm(cal2Sdf)
## Extract the elapsed time since the second to last and
## third to last for neuron 1. Normalise the result.
n1.cal2sDF[c("rlN.1","rsN.1","rtN.1")] <- brt4df(n1.cal2sDF,"lN.1",2,c("rlN.1","rsN.1","r
## load mgcv library
library(mgcv)
## fit a model with a tensorial product involving the last
## three spikes and using a cubic spline basis for the last two
## To gain time use a fixed df regression spline
n1S.fitA <- gam(event ~ te(rlN.1,rsN.1,bs="cr",fx=TRUE) + rtN.1,data=n1.cal2sDF,family=bi
## transform time
N1.Lambda <- transformedTrain(n1S.fitA)
## check out the resulting spike train using the fact
## that transformedTrain objects inherit from spikeTrain
## objects
N1.Lambda
## Use more formal checks
summary(N1.Lambda)
plot(N1.Lambda,which=c(1,2,4,5),ask=FALSE)
## Transform spike trains of neuron 2 and 3
N2.Lambda <- transformedTrain(n1S.fitA,n2.cal2sDF$event)
N3.Lambda <- transformedTrain(n1S.fitA,n3.cal2sDF$event)
```

```
## Check interactions
summary(N2.Lambda %frt% N1.Lambda)
summary(N3.Lambda %frt% N1.Lambda)
plot(N2.Lambda %frt% N1.Lambda,ask=FALSE)
plot(N3.Lambda %frt% N1.Lambda,ask=FALSE)
## End(Not run)
```

trainWAV *Generate wav Files from Spike Train(s)*

Description

Given one or two `spikeTrain` object(s), `trainWav` generates a `wav` file containing the result of the convolution of the spike train(s) with the single period of a sine function.

Usage

```
trainWAV(leftCh, rightCh = NULL, filename, leftChFreq = 500,
         rightChFreq = 1000, rate = 10000, bits = 8, pan = 50,
         overwrite = FALSE)
```

Arguments

`leftCh`	a `spikeTrain` object or a numeric vector which can be coerced to such an object. The left channel of the resulting file if two channels are specified.
`rightCh`	a `spikeTrain` object or a numeric vector which can be coerced to such an object or `NULL` (default). The right channel of the resulting file if specified and not `NULL`.
`filename`	see `saveSample`.
`leftChFreq`	the frequency of the sine function convolved with `leftCh`.
`rightChFreq`	the frequency of the sine function convolved with `rightCh`.
`rate`	see `as.Sample`.
`bits`	see `as.Sample`.
`pan`	see `stereo`.
`overwrite`	see `saveSample`.

Details

The `spikeTrain` object(s) of `leftCh` and `rightCh` are viewed as sequence of Dirac delta function and are convovled with a single period of a sine function of a given frequency.

Value

Nothing is returned, the function is used for its side effect: a `wav` file is created.

Note

You have to install the `sound` package to use these functions.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

See Also

`as.spikeTrain`, `as.Sample`, `stereo`, `saveSample`

Examples

```
## Not run:
## load spontaneous data of 4 putative projection neurons
## simultaneously recorded from the cockroach (Periplaneta
## americana) antennal lobe
data(CAL1S)
## convert data into spikeTrain objects
CAL1S <- lapply(CAL1S,as.spikeTrain)
## write train of neuron 1 to disk
trainWAV(CAL1S[[1]],,"n1spont.wav")
## write trains of neuron 1 and 2 to disk
trainWAV(CAL1S[[1]],CAL1S[[2]],"n1n2spont.wav")
## End(Not run)
```

`transformedTrain` *Performs Time Transformation of Spike Trains Fitted with glm or gam*

Description

Transform spike times from a `glm` or `gam` fitted model as defined by Ogata (1988) and Brown et al (2002). If the model structure is "correct" and if the model parameters are properly estimated the result of the time transformation should be the realization of a Poisson process with rate 1.

Usage

```
transformedTrain(obj, target = obj$data$event, select)
```

Arguments

`obj`	An object returned by `gam` or `glm`.
`target`	A binary (0,1) vector of integers with the same length as `dim(obj$data)[1]` or a vector of indexes giving the discretized times of events. All these indexes should then be included in `seq(dim(obj$data)[1])`.
`select`	A character string defining a condition to be fulfilled by the event in order to be selected, like: `time <= 6`. This is evaluated after parsing in the data frame of `obj`.

Details

The `fitted.values` component of `obj` contains the (estimated) probability to observe a spike in each time bin where the covariates required by the fitted model were defined. It is then straightforward to show using the concept of *product integral* (Kalbfleisch and Prentice, 2002; Andersen et al, 1993),provided that the time bin width is small enough to have a very small probability in each bin, that the cumulated sum of these probabilities is the expected number of events observed up to a given time. This expected number of events which is returned by `transformedTrain`. It is also the result of the "time transformation" proposed by Ogata (1988) and brought to the spike train analysis field under the name "time rescaling (theorem)" by Brown et al (2002).

`transformedTrain` can also be used to transform the times of the spikes of neurons whose spike trains were simultaneously recorded and discretized *in exactly the same way* as the neuron used to generate `obj`. This is useful to explore the possibility of functional interactions between a putative pre-synaptic neuron (whose spike train would correspond to argument `target`) and a post-synaptic one used to generate `obj`.

Value

`transformedTrain` returns an object of class `transformedTrain` inheriting from class `spikeTrain`. The object is fundamentally a numeric vector with strictly increasing elements containing the transformed times (or the expected number of events).

Note

As mentioned only the spikes for which the covariates of the model are available have their times transformed. That practically means that the length of the `transformedTrain` object returned by function `transformedTrain` *can be shorter* than the length of the original `spikeTrain` object (or more precisely than the number of spikes defined in `target`). If one works with a model involving the elapsed times since the last three spikes then the fourth spike of the train will be the first to be transformed. You should therefore expect some left truncation of the data at the beginning of each acquisition epoch.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Ogata, Yosihiko (1988) Statistical Models for Earthquake Occurrences and Residual Analysis for Point Processes. *Journal of the American Statistical Association* **83**: 9-27.

Brown, E. N., Barbieri, R., Ventura, V., Kass, R. E. and Frank, L. M. (2002) The time-rescaling theorem and its application to neural spike train data analysis. *Neural Computation* **14**: 325-346.

Kalbfleisch, John D. and Prentice, Ross L. (2002) *The Statistical Analysis of Failure Time Data.* Wiley Interscience.

Andersen, Per Kragh, Borgan, Ornulf, Gill, Richard D. and Keiding, Niels (1993) *Statistical Models Based on Counting Processes.* Springer-Verlag.

See Also

`plot.transformedTrain`, `summary.transformedTrain`, `mkGLMdf`, `data.frame`, `glm`, `mgcv`

Examples

```
## Not run:
## Let us consider neuron 1 of the CAL2S data set
data(CAL2S)
CAL2S <- lapply(CAL2S,as.spikeTrain)
CAL2S[["neuron 1"]]
renewalTestPlot(CAL2S[["neuron 1"]])
summary(CAL2S[["neuron 1"]])
## Make a data frame with a 4 ms time resolution
cal2Sdf <- mkGLMdf(CAL2S,0.004,0,60)
## keep the part relative to neuron 1, 2 and 3 separately
n1.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="1",]
n2.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="2",]
n3.cal2sDF <- cal2Sdf[cal2Sdf$neuron=="3",]
## remove unnecessary data
rm(cal2Sdf)
## Extract the elapsed time since the second to last and
## third to last for neuron 1. Normalise the result.
n1.cal2sDF[c("rlN.1","rsN.1","rtN.1")] <- brt4df(n1.cal2sDF,"lN.1",2,c("rlN.1","rsN.1","r
## load mgcv library
library(mgcv)
## fit a model with a tensorial product involving the last
## three spikes and using a cubic spline basis for the last two
## To gain time use a fixed df regression spline
n1S.fitA <- gam(event ~ te(rlN.1,rsN.1,bs="cr",fx=TRUE) + rtN.1,data=n1.cal2sDF,family=bi
## transform time
N1.Lambda <- transformedTrain(n1S.fitA)
## check out the resulting spike train using the fact
## that transformedTrain objects inherit from spikeTrain
## objects
N1.Lambda
## Use more formal checks
summary(N1.Lambda)
plot(N1.Lambda,which=c(1,2,4,5),ask=FALSE)
## Transform spike trains of neuron 2 and 3
N2.Lambda <- transformedTrain(n1S.fitA,n2.cal2sDF$event)
N3.Lambda <- transformedTrain(n1S.fitA,n3.cal2sDF$event)
## Check interactions
summary(N2.Lambda %frt% N1.Lambda)
summary(N3.Lambda %frt% N1.Lambda)
plot(N2.Lambda %frt% N1.Lambda,ask=FALSE)
plot(N3.Lambda %frt% N1.Lambda,ask=FALSE)
## End(Not run)
```

`varianceTime` *Variance-Time Analysis for Spike Trains*

Description

Performs Variance-Time Analysis for a Spike Train (or any univariate time series) assuming a Poisson Process with the same Rate as the Spike Train.

Usage

```
varianceTime(spikeTrain, CI = c(0.95, 0.99), windowSizes)
is.varianceTime(obj)
## S3 method for class 'varianceTime':
plot(x, style = c("default", "Ogata"),
     unit = "s", xlab, ylab, main, sub, xlim, ylim, ...)
```

Arguments

`spikeTrain`	a `spikeTrain` object or a vector which can be coerced to such an object.
`obj`	a object to test against a `varianceTime` object.
`x`	a `varianceTime` object.
`CI`	a numeric vector with at most two elements. The coverage probability of the confidence intervals.
`windowSizes`	a numeric increasing vector of positive numbers. The window sizes used to split the spike train.
`style`	a character. The style of the plot, `"default"` or `"Ogata"`.
`unit`	a character. The unit in which the spike times are expressed.
`xlab`	a character. The x label.
`ylab`	a character. The y label.
`main`	a character. The title.
`sub`	a character. The subtitle.
`xlim`	a numeric. See `plot`.
`ylim`	a numeric. See `plot`.
`...`	see `plot`.

Details

See Fig. 5 of Ogata (1988) for details. The confidence intervals are obtained with a Normal approximation of the Poisson distribution.

Value

`varianceTime` returns a list of class `varianceTime` with the following elements:

`s2`	numeric vector of empirical variance.
`sigma2`	numeric vector of expected variance under the Poisson hypothesis.
`ciUp`	a numeric vector or a 2 rows matrix with the upper limits of the confidence interval(s).
`ciLow`	a numeric vector or a 2 rows matrix with the lower limits of the confidence interval(s).
`windowSizes`	numeric vector of window sizes actually used.
`CI`	a numeric vector, the coverage probabilities of the confidence intervals.
`call`	the matched call

`plot.varianceTime` is used for its side effect: a graph is produced.

`is.varianceTime` returns `TRUE` if its argument is a `varianceTime` object and `FALSE` otherwise.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Ogata, Yosihiko (1988) Statistical Models for Earthquake Occurrences and Residual Analysis for Point Processes. *Journal of the American Statistical Association* **83**: 9-27.

See Also

`acf.spikeTrain`, `renewalTestPlot`

Examples

```
## Replicate (almost) Fig. 5 of Ogata 1988
data(ShallowShocks)
vtShallow <- varianceTime(ShallowShocks$Date,,c(5,10,20,40,60,80,seq(100,500,by = 25))*10
is.varianceTime(vtShallow)
plot(vtShallow, style="Ogata")
```

`weibullMLE` *Maximum Likelihood Parameter Estimation of a Weibull Model with Possibly Censored Data*

Description

Estimate Weibull model parameters by the maximum likelihood method using possibly censored data.

Usage

```
weibullMLE(yi, ni = numeric(length(yi)) + 1,
           si = numeric(length(yi)) + 1, shape.min = 0.05, shape.max = 5)
```

Arguments

`yi`	vector of (possibly binned) observations or a `spikeTrain` object.
`ni`	vector of counts for each value of `yi`; default: `numeric(length(yi))+1`.
`si`	vector of counts of *uncensored* observations for each value of `yi`; default: `numeric(length(yi))+1`.
`shape.min`	numeric, the inital guess of the minimal possible value of the `shape` parameter, used by `optimise`.
`shape.max`	numeric, the inital guess of the maximal possible value of the `shape` parameter, used by `optimise`.

Details

There is no closed form expression for the MLE of a Weibull distribution. The numerical method implemented here uses the profile likelihood described by Kalbfleisch (1985) pp 56-58.

In order to ensure good behavior of the numerical optimization routines, optimization is performed on the log of the parameters (`shape` and `scale`).

Standard errors are obtained from the inverse of the observed information matrix at the MLE. They are transformed to go from the log scale used by the optimization routine to the parameterization requested.

Value

A list of class `durationFit` with the following components:

`estimate`	the estimated parameters, a named vector.
`se`	the standard errors, a named vector.
`logLik`	the log likelihood at maximum.
`r`	a function returning the log of the relative likelihood function.
`mll`	a function returning the opposite of the log likelihood function using the log of the parameters.
`call`	the matched call.

Note

The returned standard errors (component `se`) are valid in the asymptotic limit. You should plot contours using function `r` in the returned list and check that the contours are reasonably close to ellipses.

Author(s)

Christophe Pouzat ⟨christophe.pouzat@gmail.com⟩

References

Kalbfleisch, J. G. (1985) *Probability and Statistical Inference. Volume 2: Statistical Inference*. Springer-Verlag.

Lindsey, J.K. (2004) *Introduction to Applied Statistics: A Modelling Approach*. OUP.

See Also

`Weibull`, `invgaussMLE`, `lnormMLE`, `gammaMLE`

Examples

```
## Not run:
## Simulate sample of size 100 from a weibull distribution
set.seed(1102006,"Mersenne-Twister")
sampleSize <- 100
shape.true <- 2.5
scale.true <- 0.085
sampWB <- rweibull(sampleSize,shape=shape.true,scale=scale.true)
sampWBmleWB <- weibullMLE(sampWB)
rbind(est = sampWBmleWB$estimate,se = sampWBmleWB$se,true = c(shape.true,scale.true))
```

```
## Estimate the log relative likelihood on a grid to plot contours
Shape <- seq(sampWBmleWB$estimate[1]-4*sampWBmleWB$se[1],
               sampWBmleWB$estimate[1]+4*sampWBmleWB$se[1],
               sampWBmleWB$se[1]/10)
Scale <- seq(sampWBmleWB$estimate[2]-4*sampWBmleWB$se[2],
             sampWBmleWB$estimate[2]+4*sampWBmleWB$se[2],
             sampWBmleWB$se[2]/10)
sampWBmleWBcontour <- sapply(Shape, function(sh) sapply(Scale, function(sc) sampWBmleWB$r
## plot contours using a linear scale for the parameters
## draw four contours corresponding to the following likelihood ratios:
##  0.5, 0.1, Chi2 with 2 df and p values of 0.95 and 0.99
X11(width=12,height=6)
layout(matrix(1:2,ncol=2))
contour(Shape,Scale,t(sampWBmleWBcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels=c("log(0.5)",
          "log(0.1)",
          "-1/2*P(Chi2=0.95)",
          "-1/2*P(Chi2=0.99)"),
        xlab="shape",ylab="scale",
        main="Log Relative Likelihood Contours"
        )
points(sampWBmleWB$estimate[1],sampWBmleWB$estimate[2],pch=3)
points(shape.true,scale.true,pch=16,col=2)
## The contours are not really symmetrical about the MLE we can try to
## replot them using a log scale for the parameters to see if that improves
## the situation
contour(log(Shape),log(Scale),t(sampWBmleWBcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels="",
        xlab="log(shape)",ylab="log(scale)",
        main="Log Relative Likelihood Contours",
        sub="log scale for the parameters")
points(log(sampWBmleWB$estimate[1]),log(sampWBmleWB$estimate[2]),pch=3)
points(log(shape.true),log(scale.true),pch=16,col=2)

## make a parametric boostrap to check the distribution of the deviance
nbReplicate <- 10000
sampleSize <- 100
system.time(
            devianceWB100 <- replicate(nbReplicate,{
              sampWB <- rweibull(sampleSize,shape=shape.true,scale=scale.true)
              sampWBmleWB <- weibullMLE(sampWB)
              -2*sampWBmleWB$r(shape.true,scale.true)
            }
                                       )
            )[3]

## Get 95 and 99% confidence intervals for the QQ plot
ci <- sapply(1:nbReplicate,
                 function(idx) qchisq(qbeta(c(0.005,0.025,0.975,0.995),
                                             idx,
                                             nbReplicate-idx+1),
                                       df=2)
             )
## make QQ plot
```

```
X <- qchisq(ppoints(nbReplicate),df=2)
Y <- sort(devianceWB100)
X11()
plot(X,Y,type="n",
     xlab=expression(paste(chi[2]^2," quantiles")),
     ylab="MC quantiles",
     main="Deviance with true parameters after ML fit of gamma data",
     sub=paste("sample size:", sampleSize,"MC replicates:", nbReplicate)
     )
abline(a=0,b=1)
lines(X,ci[1,],lty=2)
lines(X,ci[2,],lty=2)
lines(X,ci[3,],lty=2)
lines(X,ci[4,],lty=2)
lines(X,Y,col=2)
## End(Not run)
```

Index