

# Package ‘STAR’

January 15, 2012

**Type** Package

**Title** Spike Train Analysis with R

**Version** 0.3-4

**Date** 2009-09-29

**Depends** survival, mgcv, R2HTML, gss, codetools

**Suggests** lattice, HiddenMarkov, snow, rstream

**Author** Christophe Pouzat

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

**Description** Functions to analyze neuronal spike trains from a single neuron or from several neurons recorded simultaneously.

**License** GPL (>= 2)

**URL** <http://sites.google.com/site/spiketrainanalysiswithr>

**Repository** CRAN

**Date/Publication** 2009-09-29 16:43:46

## R topics documented:

STAR-package	3
acf.spikeTrain	3
as.repeatedTrain	5
as.spikeTrain	6
brt4df	7
changeScale	9
cockroachAIData	10
coef.durationFit	15
compModels	16
crossGeneral	18

df4counts	25
diff.spikeTrain	27
dinvgauss	28
dllogis	31
dexp	33
frt	35
gamlockedTrain	36
gammaMLE	39
gamObj	42
gampsth	43
gsslockedTrain	45
gssObj	48
gsspsth	49
hgamma	53
hist.lockedTrain	57
invgaussMLE	60
isi	68
isiHistFit	70
jpsth	71
llogisMLE	74
lnormMLE	77
lockedTrain	80
mkAR	83
mkCPSP	85
mkDummy	87
mkGLMdf	88
mkM2U	94
mkREdf	96
plot.frt	98
plot.quickPredict	100
plot.spikeTrain	102
plot.ssanova	104
plot.transformedTrain	105
predictLogProb	108
print.repeatedTrain	110
print.spikeTrain	112
psth	113
purkinjeCellData	116
qqDuration	117
quickPredict	119
raster	120
rateEvolution	122
renewalTestPlot	124
reportHTML	126
reportHTML.gam	127
reportHTML.repeatedTrain	128
reportHTML.spikeTrain	130
rexpMLE	133

ShallowShocks . . . . . 134  
 summary.CountingProcessSamplePath . . . . . 136  
 summary.transformedTrain . . . . . 145  
 thinProcess . . . . . 147  
 transformedTrain . . . . . 156  
 varianceTime . . . . . 158  
 weibullMLE . . . . . 160  
 %tt% . . . . . 163

**Index** **166**

STAR-package *Spike Train Analysis with R*

**Description**

Functions to analyze neuronal spike trains

**Details**

Package: STAR  
 Type: Package  
 Version: 0.2-1  
 Date: 2008-11-02  
 Depends: gss, survival, mgcv, R2HTML  
 Suggests: lattice, HiddenMarkov, snow, rstream  
 License: GPL version 2 or newer  
 URL: <http://sites.google.com/site/spiketrainanalysiswithr>

**Author(s)**

Christophe Pouzat  
 Maintainer: Christophe Pouzat <[christophe.pouzat@gmail.com](mailto:christophe.pouzat@gmail.com)>

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 object 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 elements 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 <code>data.frame</code> generated by <code>mkGLMdf</code> 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, 1N.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 <code>data.frame</code> . 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 <code>function</code> 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, Val'erie (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("r1N.1","rsN.1","rtN.1")] <- brt4df(n1.cal2sDF,"1N.1",2,c("r1N.1","rsN.1","rtN.1"))
## 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(r1N.1,rsN.1,bs="cr") + rtN.1,data=n1.cal2sDF,family=binomial(link="logit"))
summary(n1S.fitA)
## plot the result in 2 different ways
plot(n1S.fitA)
vis.gam(n1S.fitA,phi=20,theta=45)

## End(Not run)

```

---

changeScale

*Change the Scales of a quickPredict Object for an Interaction Term*


---

## Description

Designed to transform results of [quickPredict](#) obtained on interaction terms from the transformed scale (on which the variables are approximately uniformly distributed) onto the "native", linear scale.

## Usage

```
changeScale(obj, xFct, yFct)
```

## Arguments

obj	a <a href="#">quickPredict</a> object.
xFct	a function to be applied on the xx element of obj. This function should be the qFct attribute of the function, returned by <a href="#">mkM2U</a> , used to transform the variable from the "native" to the "uniform" scale.
yFct	a function to be applied on the yy element of obj. This function should be the qFct attribute of the function, returned by <a href="#">mkM2U</a> , used to transform the variable from the "native" to the "uniform" scale.

## Value

A [quickPredict](#) object.

**Author(s)**

Christophe Pouzat <christophe.pouzat@gmail.com>

**See Also**

[quickPredict](#), [plot.quickPredict](#)

**Examples**

```
## Not run:
data(e060824spont)
DFA <- subset(mkGLMdf(e060824spont,0.004,0,59),neuron==1)
DFA <- within(DFA,i1 <- isi(DFA,lag=1))
DFA <- DFA[complete.cases(DFA),]
m2u1 <- mkM2U(DFA,"1N.1",0,29)
m2ui <- mkM2U(DFA,"i1",0,29,maxiter=200)
DFA <- within(DFA,e1t <- m2u1(1N.1))
DFA <- within(DFA,i1t <- m2ui(i1))
with(DFA,plot(ecdf(e1t[time>29]),pch="."))
abline(a=0,b=1,col=2,lty=2)
with(DFA,plot(ecdf(i1t[time>29]),pch="."))
abline(a=0,b=1,col=2,lty=2)
m1.fit <- gssanova(event~e1t*i1t, data=subset(DFA,time>29), family="binomial", seed=20061001)
inter.pred <- m1.fit %qp% "e1t:i1t"
contour(inter.pred,what="mean",nlevels=10,col=2,lwd=2)
contour(inter.pred,what="sd",nlevels=5,col=1,lwd=1,lty=2,add=TRUE)
inter.predN <- changeScale(inter.pred,attr(m2u1,"qFct"),attr(m2ui,"qFct"))
contour(inter.predN,what="mean",nlevels=5,col=2,lwd=1)
contour(inter.predN,what="sd",nlevels=3,col=1,lwd=1,lty=2,add=TRUE)

## End(Not run)
```

---

cockroachAIData

*Spike Trains of several Cockroach Antennal Lobe Neurons Recorded from Six Animals*

---

**Description**

Four (CAL1S and CAL1V), three (CAL2S and CAL2C), three (e060517spont and e060517ionon), three (e060817spont, e060817terpi, e060817citron and e060817mix), two (e060824spont and e060824citra1) and four (e070528spont and e070528citronellal) Cockroach (*Periplaneta americana*) antennal lobe neurons (putative projection neurons) were recorded simultaneously and extracellularly during spontaneous activity and odors (vanilin, citral, citronellal, terpineol, beta-ionon) responses from six different animals. The data sets contain the sorted spike trains of the neurons.

**Usage**

```

data(CAL1S)
data(CAL1V)
data(CAL2S)
data(CAL2C)
data(e060517spont)
data(e060517ionon)
data(e060817spont)
data(e060817terpi)
data(e060817citron)
data(e060817mix)
data(e060824spont)
data(e060824citrall)
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 *vanillin* (<http://en.wikipedia.org/wiki/Vanillin>). 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 *citrall* (<http://en.wikipedia.org/wiki/Citrall>). Each acquisition was 14 s long. The command to the odor delivery valve was on between sec 5.87 and sec 6.37.

e060517spont is a named list of with 3 components ("neuron 1", "neuron 2", "neuron 3"). Each component is a spikeTrain object (ie, action potentials occurrence times) of one neuron recorded during 61 s of spontaneous activity. *Times are expressed in seconds.*

e060517ionon is a named list with 3 components ("neuron 1", "neuron 2", "neuron 3"). Each component is a repeatedTrain object with 19 spikeTrain objects: "stim. 1", ..., "stim. 19". Each spikeTrain contains the spike train of one neuron during 1 stimulation (odor puff) with *beta-ionon* (<http://commons.wikimedia.org/wiki/Image:Beta-Ionon.svg>). Each acquisition was 15 s long. The command to the odor delivery valve was on between sec 6.07 and sec 6.57.

e060817spont is a named list of with 3 components ("neuron 1", "neuron 2", "neuron 3"). 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.*

e060817terpi is a named list with 3 components ("neuron 1", "neuron 2", "neuron 3"). Each component is a repeatedTrain object with 20 spikeTrain objects: "stim. 1", ..., "stim. 20". Each spikeTrain contains the spike train of one neuron during 1 stimulation (odor puff) with *terpineol* (<http://en.wikipedia.org/wiki/Terpineol>). Each acquisition was 15 s long. The command to the odor delivery valve was on between sec 6.03 and sec 6.53.

e060817citron is a named list with 3 components ("neuron 1", "neuron 2", "neuron 3"). Each component is a repeatedTrain object with 20 spikeTrain objects: "stim. 1", ..., "stim. 20". Each spikeTrain contains the spike train of one neuron during 1 stimulation (odor puff) with *citronellal* (<http://en.wikipedia.org/wiki/Citronellal>). Each acquisition was 15 s long. The command to the odor delivery valve was on between sec 5.99 and sec 6.49.

e060817mix is a named list with 3 components ("neuron 1", "neuron 2", "neuron 3"). Each component is a repeatedTrain object with 20 spikeTrain objects: "stim. 1", ..., "stim. 20". Each spikeTrain contains the spike train of one neuron during 1 stimulation (odor puff) with a mixture of *terpinaol* and *citronellal* (the sum of the two previous stim.). Each acquisition was 15 s long. The command to the odor delivery valve was on between sec 6.01 and sec 6.51.

e060824spont is a named list of with 2 components ("neuron 1", "neuron 2"). Each component is a spikeTrain object (ie, action potentials occurrence times) of one neuron recorded during 59 s of spontaneous activity. *Times are expressed in seconds.*

e060824citrals is a named list with 2 components ("neuron 1", "neuron 2"). 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 *citrals* (<http://en.wikipedia.org/wiki/Citrals>). Each acquisition was 15 s long. The command to the odor delivery valve was on between sec 6.01 and sec 6.51.

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.*

e070528citronellals 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* (<http://en.wikipedia.org/wiki/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

Every repeatedTrain object of these data sets has an attribute named `stimTimeCourse` containing the opening and closing times of the odor delivery valve.

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 artifacts on cross-correlograms.

## Source

Recording and spike sorting performed by Antoine Chaffiol <[antoine.chaffiol@univ-paris5.fr](mailto:antoine.chaffiol@univ-paris5.fr)> at the Cerebral Physiology Lab, CNRS UMR 8118: [http://www.biomedicale.univ-paris5.fr/phycerv/physiologie\\_cerebrale.htm](http://www.biomedicale.univ-paris5.fr/phycerv/physiologie_cerebrale.htm).

## References

[http://www.biomedicale.univ-paris5.fr/physcerv/C\\_Pouzat/Doc/ChaffiolEtAl\\_FENS2006.pdf](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 1st 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)
## Get the stimulus time course
attr(e070528citronellal[["neuron 1"]],"stimTimeCourse")
## look at the raster of the 1st neuron
plot(e070528citronellal[["neuron 1"]],stim=c(6.14,6.64))

## Not run:
## A "detailed" analysis of e060817 were 2 odors as well as there mixtures
## were used.
## Load the terpeneol, citronellal and mixture response data
data(e060817terpi)
data(e060817citron)
data(e060817mix)
## get smooth psth with gsspsth0
e060817terpiN1PSTH <- gsspsth0(e060817terpi[["neuron 1"]])
e060817terpiN2PSTH <- gsspsth0(e060817terpi[["neuron 2"]])
e060817terpiN3PSTH <- gsspsth0(e060817terpi[["neuron 3"]])
e060817citronN1PSTH <- gsspsth0(e060817citron[["neuron 1"]])
e060817citronN2PSTH <- gsspsth0(e060817citron[["neuron 2"]])
e060817citronN3PSTH <- gsspsth0(e060817citron[["neuron 3"]])
e060817mixN1PSTH <- gsspsth0(e060817mix[["neuron 1"]])
```

```

e060817mixN2PSTH <- gsspsth0(e060817mix[["neuron 2"]])
e060817mixN3PSTH <- gsspsth0(e060817mix[["neuron 3"]])
## look at them
## Neuron 1
plot(e060817terpiN1PSTH,stimTimeCourse=attr(e060817terpi[["neuron 1"]], "stimTimeCourse"),colCI=2)
plot(e060817citronN1PSTH,stimTimeCourse=attr(e060817citron[["neuron 1"]], "stimTimeCourse"),colCI=2)
plot(e060817mixN1PSTH,stimTimeCourse=attr(e060817mix[["neuron 1"]], "stimTimeCourse"),colCI=2)
## Neuron 2
plot(e060817terpiN2PSTH,stimTimeCourse=attr(e060817terpi[["neuron 2"]], "stimTimeCourse"),colCI=2)
plot(e060817citronN2PSTH,stimTimeCourse=attr(e060817citron[["neuron 2"]], "stimTimeCourse"),colCI=2)
plot(e060817mixN2PSTH,stimTimeCourse=attr(e060817mix[["neuron 2"]], "stimTimeCourse"),colCI=2)
## Neuron 3
plot(e060817terpiN3PSTH,stimTimeCourse=attr(e060817terpi[["neuron 3"]], "stimTimeCourse"),colCI=2)
plot(e060817citronN3PSTH,stimTimeCourse=attr(e060817citron[["neuron 3"]], "stimTimeCourse"),colCI=2)
plot(e060817mixN3PSTH,stimTimeCourse=attr(e060817mix[["neuron 3"]], "stimTimeCourse"),colCI=2)

## Make now fancier plots with superposed psths ####
## Take into account the fact that the stimuli onsets are not identical

## Neuron 1
plot(e060817mixN1PSTH$mids-0.02,e060817mixN1PSTH$ciUp,type="n",ylim=c(0,max(e060817mixN1PSTH$ciUp)),xlim=c(5,1),
rect(5.99,0,6.49,max(e060817mixN1PSTH$ciUp),col="grey80",border=NA)
abline(h=0)
polygon(c(e060817mixN1PSTH$mids-0.02,rev(e060817mixN1PSTH$mids-0.02)),c(e060817mixN1PSTH$ciLow,rev(e060817mixN1PSTH$ciLow)),col="grey80",border=NA)
polygon(c(e060817citronN1PSTH$mids,rev(e060817citronN1PSTH$mids)),c(e060817citronN1PSTH$ciLow,rev(e060817citronN1PSTH$ciLow)),col="grey80",border=NA)
polygon(c(e060817terpiN1PSTH$mids-0.04,rev(e060817terpiN1PSTH$mids-0.04)),c(e060817terpiN1PSTH$ciLow,rev(e060817terpiN1PSTH$ciLow)),col="grey80",border=NA)
lines(e060817terpiN1PSTH$mids-0.04,e060817terpiN1PSTH$freq,col=rgb(0,0,1),lwd=2)
lines(e060817citronN1PSTH$mids,e060817citronN1PSTH$freq,col=rgb(1,0,0),lwd=2)
lines(e060817mixN1PSTH$mids-0.02,e060817mixN1PSTH$freq,col=rgb(0,0,0),lwd=2)
legend(8,0.9*max(e060817mixN1PSTH$ciUp),c("Terpineol","Citronellal","Mixture"),col=c(4,2,1),lwd=2)

## Neuron 2
plot(e060817mixN2PSTH$mids-0.02,e060817mixN2PSTH$ciUp,type="n",ylim=c(0,max(e060817mixN2PSTH$ciUp)),xlim=c(5,1),
rect(5.99,0,6.49,max(e060817mixN2PSTH$ciUp),col="grey80",border=NA)
abline(h=0)
polygon(c(e060817mixN2PSTH$mids-0.02,rev(e060817mixN2PSTH$mids-0.02)),c(e060817mixN2PSTH$ciLow,rev(e060817mixN2PSTH$ciLow)),col="grey80",border=NA)
polygon(c(e060817citronN2PSTH$mids,rev(e060817citronN2PSTH$mids)),c(e060817citronN2PSTH$ciLow,rev(e060817citronN2PSTH$ciLow)),col="grey80",border=NA)
polygon(c(e060817terpiN2PSTH$mids-0.04,rev(e060817terpiN2PSTH$mids-0.04)),c(e060817terpiN2PSTH$ciLow,rev(e060817terpiN2PSTH$ciLow)),col="grey80",border=NA)
lines(e060817terpiN2PSTH$mids-0.04,e060817terpiN2PSTH$freq,col=rgb(0,0,1),lwd=2)
lines(e060817citronN2PSTH$mids,e060817citronN2PSTH$freq,col=rgb(1,0,0),lwd=2)
lines(e060817mixN2PSTH$mids-0.02,e060817mixN2PSTH$freq,col=rgb(0,0,0),lwd=2)
legend(8,0.9*max(e060817mixN2PSTH$ciUp),c("Terpineol","Citronellal","Mixture"),col=c(4,2,1),lwd=2)

## Neuron 3
plot(e060817mixN3PSTH$mids-0.02,e060817mixN3PSTH$ciUp,type="n",ylim=c(0,max(e060817mixN3PSTH$ciUp)),xlim=c(5,1),
rect(5.99,0,6.49,max(e060817mixN3PSTH$ciUp),col="grey80",border=NA)
abline(h=0)
polygon(c(e060817mixN3PSTH$mids-0.02,rev(e060817mixN3PSTH$mids-0.02)),c(e060817mixN3PSTH$ciLow,rev(e060817mixN3PSTH$ciLow)),col="grey80",border=NA)
polygon(c(e060817citronN3PSTH$mids,rev(e060817citronN3PSTH$mids)),c(e060817citronN3PSTH$ciLow,rev(e060817citronN3PSTH$ciLow)),col="grey80",border=NA)
polygon(c(e060817terpiN3PSTH$mids-0.04,rev(e060817terpiN3PSTH$mids-0.04)),c(e060817terpiN3PSTH$ciLow,rev(e060817terpiN3PSTH$ciLow)),col="grey80",border=NA)
lines(e060817terpiN3PSTH$mids-0.04,e060817terpiN3PSTH$freq,col=rgb(0,0,1),lwd=2)
lines(e060817citronN3PSTH$mids,e060817citronN3PSTH$freq,col=rgb(1,0,0),lwd=2)
lines(e060817mixN3PSTH$mids-0.02,e060817mixN3PSTH$freq,col=rgb(0,0,0),lwd=2)

```

```
legend(8,0.9*max(e060817mixN3PSTH$ciUp),c("Terpineol","Citronellal","Mixture"),col=c(4,2,1),lwd=2)

## End(Not run)
```

---

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 1st 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

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

## Details

Fits are performed by maximizing the likelihood.

**Value**

A vector whose component are named 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)

```

---

crossGeneral

*Computations of Boundary Crossing Probabilities for the Wiener Process*


---

## Description

Computes the distribution of the first passage time through an arbitrary (crossGeneral) or a "tight" (crossTight) boundary for a Wiener process. The method of Loader and Deely (1987) is used. A tight boundary is a boundary generating the tightest confidence band for the process (Kendall et al, 2007). Utility function and methods: mkTightBMtargetFct, print, summary, plot, lines, are also provided to use and explore the results.

## Usage

```

crossGeneral(tMax = 1, h = 0.001, cFct, cprimeFct, bFct, withBounds = FALSE, Lplus)
crossTight(tMax = 1, h = 0.001, a = 0.3, b = 2.35, withBounds = TRUE, logScale = FALSE)
mkTightBMtargetFct(ci = 0.95, tMax = 1, h = 0.001, logScale = FALSE)
## S3 method for class 'FirstPassageTime'
print(x, ...)
## S3 method for class 'FirstPassageTime'
summary(object, digits, ...)
## S3 method for class 'FirstPassageTime'
plot(x, y, which = c("Distribution", "density"), xlab, ylab, ...)
## S3 method for class 'FirstPassageTime'
lines(x, which = c("Distribution", "density"), ...)

```

## Arguments

tMax	A positive numeric. The "time" during which the Wiener process is followed.
h	A positive numeric. The integration time step used for the numerical solution of the Volterra integral equation (see details).
cFct	A function defining the boundary to be crossed. The first argument of the function should be a "time" argument. If the first argument is a vector, the function should return a vector of the same length.

cprimeFct	A function defining time derivative of the boundary to be crossed. Needs to be specified only if a check of the sign of the kernel derivative (see details) is requested. The first argument of the function should be a "time" argument. If the first argument is a vector, the function should return a vector of the same length.
bFct	A function. The "b" function of Loader and Deely (1987). Does not need to be specified ( <i>i.e.</i> , can be missing) but can be used to improve convergence. The first argument of the function should be a "time" argument. If the first argument is a vector, the function should return a vector of the same length.
withBounds	A logical. Should bounds on the distribution be calculated? If yes, set it to TRUE, leave it to its default value, FALSE, otherwise.
Lplus	A logical. If bounds are requested (withBounds=TRUE) and if the sign of the time derivative of the kernel is known to be positive or null, set to TRUE, if it is known to be negative, set it to FALSE. If the sign is unknown, leave Lplus unspecified and provide a cprimeFct function.
logScale	A logical. Should intermediate calculations in crossTight be carried out on the log scale for numerical precision? If yes, set it to TRUE, leave it to its default, FALSE, otherwise.
a,b	numerics, the two parameters of the "tight" boundary: $c(t) = a + b\sqrt{t}$ . See details.
ci	A numeric larger than 0 and smaller than 1. The nominal coverage probability desired for a "tight" confidence band (see details).
x,object	A FirstPassageTime object returned by crossGeneral or crossTight.
y	Not used but required for a plot method.
which	A character string, "Distribution" or "density", specifying if a probability distribution or a probability density should be graphed.
xlab,ylab	See <a href="#">plot</a> .
digits	A positive integer. The number of digits to print in summary. If bounds were computed, the value of digits is computed internally based on the bounds width.
...	Used in plot and lines to pass further arguments (see <a href="#">plot</a> and <a href="#">lines</a> ), not used in print and summary.

## Details

The Loader and Deely (1987) method to compute the probability  $G(t)$  that the first passage of a Wiener process / Brownian motion occurs between 0 and  $t$  (argument tMax of crossGeneral and crossTight) through a boundary defined by  $c(t)$  is based on the numerical solution of a Volterra integral equation of the first kind satisfied by  $G()$  and defined by their Eq. 2.2:

$$F(t) = \int_0^t K(t,u)dG(u)$$

where,  $F(t)$  is defined by:

$$F(t) = \Phi\left(-\frac{c(t)}{\sqrt{t}}\right) + \exp\left(-2b(t)(c(t) - tb(t))\right) \Phi\left(\frac{-c(t) + 2tb(t)}{\sqrt{t}}\right)$$

$K(t, u)$  is defined by:

$$K(t, u) = \Phi\left(\frac{c(u) - c(t)}{\sqrt{t - u}}\right) + \exp(-2b(t)(c(t) - c(u) - (t - u)b(t))) \Phi\left(\frac{c(u) - c(t) + 2(t - u)b(t)}{\sqrt{t - u}}\right)$$

and  $b(t)$  is an additional function (that can be uniformly 0) that is chosen to improve convergence speed and to get error bounds. Argument  $h$  is the step size used in the numerical solution of the above Volterra integral equation. The mid-point method (Eq. 3.1 and 3.2 of Loader and Deely (1987)) is implemented. If  $tMax$  is not a multiple of  $h$  it is modified as follows:  $tMax \leftarrow \text{round}(tMax/h) * h$ .

`crossGeneral` generates functions  $F()$  and  $K(,)$  internally given  $c()$  (argument `cFct`) and  $b()$  (argument `bFct`). If `bFct` is not given (*i.e.*, `missing(bFct)` returns TRUE) it is taken as uniformly 0. If a numeric is given for `cFct` then  $c()$  is defined as a uniform function returning the first element of the argument (`cFct`).

Function `crossTight` assumes the following functional form for  $c()$ :  $c(t) = a + b\sqrt{t}$ .  $b()$  is set to  $c'()$  (the derivative of  $c()$ ). Arguments  $a$  and  $b$  of `crossTight` correspond to the 2 parameters of  $c()$ .

If argument `withBounds` is set to TRUE then bounds on  $G()$  are computed. Function `crossTight` uses Eq. 3.6 and 3.7 of Loader and Deely (1987) to compute these bounds,  $G_L(t)$  and  $G_U(t)$ . Function `crossGeneral` uses Eq. 3.6 and 3.7 (if argument `Lplus` is set to TRUE) or Eq. 3.10 and 3.11 (if argument `Lplus` is set to FALSE). Here `Lplus` stands for the sign of the partial derivative of the kernel  $K(,)$  with respect to its second argument. If the sign is not known the user can provide the derivative  $c'()$  of  $c()$  as argument `cprimeFct`. A (slow) numerical check is then performed to decide whether `Lplus` should be TRUE or FALSE or if it changes sign (in which case bounds cannot be obtained and an error is returned).

In function `crossTight` argument `logScale` controls the way some intermediate computations of the mid-point method are implemented. If set to FALSE (the default) a literal implementation of Eq. 3.2 of Loader and Deely (1987) is used. If set to TRUE then additions subtractions are computed on the log scale using functions `logspace_add` and `logspace_sub` of the R API. The computation is then slightly slower and it turns out that the gain in numerical precision is not really significant, so you can safely leave this argument to its default value.

## Value

`crossGeneral` and `crossTight` return a `FirstPassageTime` object which is a list with the following components:

<code>time</code>	A numeric vector of "times" at which the first passage time probability has been evaluated.
<code>G</code>	A numeric vector of first passage probability.
<code>Gu</code>	A numeric vector with the upper bound of first passage probability. Only if <code>withBounds</code> was set to TRUE.
<code>Gl</code>	A numeric vector with the lower bound of first passage probability. Only if <code>withBounds</code> was set to TRUE.
<code>mids</code>	A numeric vector of "times" at which the first passage time probability <i>density</i> has been evaluated. Mid points of component <code>time</code> .
<code>g</code>	A numeric vector of first passage probability <i>density</i> .

h                    A numeric. The value of argument h of crossGeneral or crossTight.  
 call                The matched call.

mkTightBMtargetFct returns a function which can be used in optim. This function returns the square of the difference between  $(1-ci)/2$  (remember the "symmetry" of the Wiener processes paths, that is, for every path there is a symmetric one with respect to the abscissa having *with the same probability*) and the probability to have the first passage time of the Wiener process smaller or equal to 1 when the boundary is the "tight" boundary defined by:  $a + b\sqrt{t}$ . The function takes a single vector argument containing the *log* of the parameters a (vector's first element) and b (vector's second element).

Methods print.FirstPassageTime and summary.FirstPassageTime output the probability to observe the first exit between 0 and tMax. If bounds were computed, the precision on the probability is also returned (as an attribute for print.FirstPassageTime). summary.FirstPassageTime also gives the integration time step, h, used.

### Warning

crossGeneral with withBounds = TRUE and a negative kernel derivative is presently poorly tested, so be careful and let me know if mistakes show up.

### Note

Using logScale = TRUE in crossTight seems to be an overkill, that is, it doubles computation's time without bringing significant numerical improvement.

crossGeneral is for now pure R code. The first passage probability is obtained by solving the lower triangular system (Eq. 3.1 of Loader and Deely (1987)) with [forwardsolve](#) and is therefore rather fast (but can be memory "hungry"). The bounds are computed by 2 nested loops and can therefore be long to compute.

crossTight is calling a C code and is fast.

Loader and Deely paper also describes a method where  $G(t)$  is solution of a Volterra integral equation of the second kind (their Eq. 2.7). This approach is presently not implemented in the above functions.

### Author(s)

Christophe Pouzat <christophe.pouzat@gmail.com>

### References

- C. R. Loader and J. J. Deely (1987) Computations of Boundary Crossing Probabilities for the Wiener Process. *J. Statist. Comput. Simul.* **27**: 95–105.
- W. S. Kendall, J.- M. Marin and C. P. Robert (2007) Brownian Confidence Bands on Monte Carlo Output. *Statistics and Computing* **17**: 1–10. Preprint available at: <http://www.ceremade.dauphine.fr/%7Exian/kmr04.rev.pdf>

### See Also

[print](#), [summary](#), [plot](#), [lines](#), [pinvgauss](#)

**Examples**

```

## Reproduce Table 1 (p 101) of Loader and Deely (1987)
## define a vector of n values
nLD <- c(8,16,32,64,128)

## Part 1: c(t) = sqrt(1+t) and tMax=1
## define cFct
cFT1p1 <- function(t) sqrt(1+t)
## define the different bFct
bFT1p1.ii <- function(t) 0.5/sqrt(1+t)
bFT1p1.iii <- function(t) (cFT1p1(t)-cFT1p1(0))/t
bFT1p1.iv <- function(t) 0.5*(bFT1p1.ii(t)+bFT1p1.iii(t))
bFT1p1.v <- function(t) (2*t-4/5*((1+t)^2.5-1))/t^3+3*cFT1p1(t)/2/t
## Do the calculations
round(t(sapply(nLD,
              function(n) {
                c(n=n,
                  i=crossGeneral(tMax=1,h=1/n,cFct=cFT1p1)$G[n],
                  ii=crossGeneral(tMax=1,h=1/n,cFct=cFT1p1,bFct=bFT1p1.ii)$G[n],
                  iii=crossGeneral(tMax=1,h=1/n,cFct=cFT1p1,bFct=bFT1p1.iii)$G[n],
                  iv=crossGeneral(tMax=1,h=1/n,cFct=cFT1p1,bFct=bFT1p1.iv)$G[n],
                  v=crossGeneral(tMax=1,h=1/n,cFct=cFT1p1,bFct=bFT1p1.v)$G[n])))),
      digits=6)

## Part 2: c(t) = exp(-t) and tMax=1
## define cFct
cFT1p2 <- function(t) exp(-t)
## define the different bFct
cFT1p2 <- function(t) exp(-t)
bFT1p2.ii <- function(t) -exp(-t)
bFT1p2.iii <- function(t) (cFT1p2(t)-cFT1p2(0))/t
bFT1p2.iv <- function(t) 0.5*(bFT1p2.ii(t)+bFT1p2.iii(t))
bFT1p2.v <- function(t) 3*(1-t-exp(-t))/t^3+3*cFT1p2(t)/2/t
## Do the calculations
round(t(sapply(nLD,
              function(n) {
                c(n=n,
                  i=crossGeneral(tMax=1,h=1/n,cFct=cFT1p2)$G[n],
                  ii=crossGeneral(tMax=1,h=1/n,cFct=cFT1p2,bFct=bFT1p2.ii)$G[n],
                  iii=crossGeneral(tMax=1,h=1/n,cFct=cFT1p2,bFct=bFT1p2.iii)$G[n],
                  iv=crossGeneral(tMax=1,h=1/n,cFct=cFT1p2,bFct=bFT1p2.iv)$G[n],
                  v=crossGeneral(tMax=1,h=1/n,cFct=cFT1p2,bFct=bFT1p2.v)$G[n])))),
      digits=6)

## Part 3: c(t) = t^2 + 3*t + 1 and tMax=1
## define cFct
cFT1p3 <- function(t) t^2+3*t+1
## define the different bFct
bFT1p3.ii <- function(t) 2*t+3
bFT1p3.iii <- function(t) (cFT1p3(t)-cFT1p3(0))/t
bFT1p3.v <- function(t) 5*t/4+3
bFT1p3.vi <- function(t) rep(3,length(t))

```

```

round(t(sapply(nLD,
              function(n) {
                c(n=n,
                  i=crossGeneral(tMax=1,h=1/n,cFct=cFT1p3)$G[n],
                  ii=crossGeneral(tMax=1,h=1/n,cFct=cFT1p3,bFct=bFT1p3.ii)$G[n],
                  iii=crossGeneral(tMax=1,h=1/n,cFct=cFT1p3,bFct=bFT1p3.iii)$G[n],
                  v=crossGeneral(tMax=1,h=1/n,cFct=cFT1p3,bFct=bFT1p3.v)$G[n],
                  vi=crossGeneral(tMax=1,h=1/n,cFct=cFT1p3,bFct=bFT1p3.vi)$G[n])))),
      digits=6)

## Part 3: c(t) = t^2 + 3*t + 1 and tMax=1
## define cFct
cFT1p4 <- function(t) 1/t
## Here only column (i) and (vii) are reproduced.
## If one attempts to reproduce (ii) directly with crossGeneral
## a NaN appears (when a -Inf is the correct value) in functions
## F(.) and K(.) (see details) for instance when t=0 in F.
## Then as crossGeneral is presently written R attempts to
## compute t*b(t), where b(t) is c'(t), that is, t*(-1/t^2) which is
## NaN (for R) when t=0.
bFT1p4.vii <- function(t) rep(-1,length(t))
round(t(sapply(nLD,
              function(n) {
                c(n=n,
                  i=crossGeneral(tMax=1,h=1/n,cFct=cFT1p4)$G[n],
                  vii=crossGeneral(tMax=1,h=1/n,cFct=cFT1p4,bFct=bFT1p4.vii)$G[n])))),
      digits=6)
## The last 3 rows of column (vii) are not the same as in the paper

## Reproduce Table 4 (p 104) of Loader and Deely (1987)
## As before the probability of first passage between
## 0 and 1 is computed. This time only three boundary
## functions are considered. The error bounds are
## obtained

## Part 1: c(t) = sqrt(1+t)
## Left columns pair: b(t) = c'(t)
round(t(sapply(nLD,
              function(n) {
                res <- crossGeneral(tMax=1,h=1/n,cFct=cFT1p1,bFct=bFT1p1.ii,withBounds=TRUE,Lplus=TRUE)
                c(Gl=res$Gl[n],Gu=res$Gu[n])
              }
            ),
      digits=5)

## Right columns pair: b(t) = 0
round(t(sapply(nLD,
              function(n) {
                res <- crossGeneral(tMax=1,h=1/n,cFct=cFT1p1,withBounds=TRUE,Lplus=TRUE)
                c(n=n,Gl=res$Gl[n],Gu=res$Gu[n])
              }
            )

```

```

    ),
    digits=5)

## Part 2: c(t) = t^2 + 3*t + 1
## Left columns pair: b(t) = 3 - 2*t
round(t(sapply(nLD,
  function(n) {
    res <- crossGeneral(tMax=1,h=1/n,cFct=cFT1p3,bFct=function(t) 3-2*t,withBounds=TRUE,Lplus=TRUE)
    c(n=n,Gl=res$Gl[n],Gu=res$Gu[n])
  }
  )
),
  digits=5)

## Right columns pair: b(t) = 3 - t
round(t(sapply(nLD,
  function(n) {
    res <- crossGeneral(tMax=1,h=1/n,cFct=cFT1p3,bFct=function(t) 3-2*t,withBounds=TRUE,Lplus=TRUE)
    c(n=n,Gl=res$Gl[n],Gu=res$Gu[n])
  }
  )
),
  digits=5)

## Part 3: c(t) = 1 + sin(t)
## Left columns pair: b(t) = c'(t)
round(t(sapply(nLD,
  function(n) {
    res <- crossGeneral(tMax=1,h=1/n,cFct=function(t) 1+sin(t),bFct=function(t) cos(t),withBounds=TRUE,L
    c(n=n,Gl=res$Gl[n],Gu=res$Gu[n])
  }
  )
),
  digits=5)

## Left columns pair: b(t) = 0.5
round(t(sapply(nLD,
  function(n) {
    res <- crossGeneral(tMax=1,h=1/n,cFct=function(t) 1+sin(t),bFct=function(t) rep(0.5,length(t)),withB
    c(n=n,Gl=res$Gl[n],Gu=res$Gu[n])
  }
  )
),
  digits=5)

## Check crossGeneral against an analytical inverse Gaussian
## distribution
## Define inverse Gaussian parameters
mu.true <- 0.075
sigma2.true <- 3
## Define a function transforming the "drift" (mu.true) and
## "noise variance" (sigma2.true) of the default inverse

```

```

## Gaussian parametrization of STAR into a
## linear boundary of an equivalent Wiener process first
## passage time problem
star2ld <- function(mu,sigma2) c(sqrt(1/sigma2),-sqrt(1/sigma2)/mu)
## Get the "equivalent" boundary parameters (y intercept and slope)
parB1 <- star2ld(mu.true,sigma2.true)
## Plot the "target" inverse Gaussian density
xx <- seq(0.001,0.3,0.001)
plot(xx,dinvgauss(xx,mu=mu.true,sigma2=sigma2.true),type="l")
## Get the numerical estimate of the density using Loader and
## Deely Volterra integral equation method
igB1 <- crossGeneral(tMax=0.3,h=0.001,cFct=function(t) parB1[1]+parB1[2]*t,withBounds=FALSE)
## superpose the numerical estimate to the exact solution
## use lines method to do that
lines(igB1,"density",col=2)

## Use of crossTight and associated function
## Get the paramters, a and b, of the "approximate"
## tightest boundary:  $c(t) = a + b\sqrt{t}$ , giving a
## 0.05 probability of exit between 0 and 1
## (in fact we are discussing here a pair of symmetrical
## bounaries,  $c(t)$  and  $-c(t)$ ). See Kendall et al (2007)
## for details
## Start by defining the target function
target95 <- mkTightBMtargetFct(ci=0.95)
## get the optimal log(a) and log(b) using
## the values of table 1 of Kendall et al as initial
## guesses
p95 <- optim(log(c(0.3,2.35)),target95,method="BFGS")
## check the convergence of BFGS
p95$convergence
## check if the parameters changed a lot
exp(p95$par)
## Get the bounds on G(1) for these optimal parameters
d95 <- crossTight(a=exp(p95$par[1]),b=exp(p95$par[2]),withBound=TRUE,logScale=FALSE)
## print out the summary
summary(d95)
## Not run:
## Do the same for the 0.01 probability of first passage
target99 <- mkTightBMtargetFct(ci=0.99)
p99 <- optim(p95$par,target99,method="BFGS")
p99$convergence
exp(p99$par)
d99 <- crossTight(a=exp(p99$par[1]),b=exp(p99$par[2]),withBound=TRUE,logScale=FALSE)
summary(d99)

## 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 <code>factor</code> ).
Trial	the trial index (a <code>factor</code> ).
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,xlim=c(0,10))
## 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 <code>length(n) &gt; 1</code> , 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 $P[X \leq x]$ , otherwise, $P[X > x]$ .
log, log.p	logical; if TRUE, probabilities p are given as $\log(p)$ .

### Details

With the default, "sigma2", parameterization ( $\mu = m$ ,  $\text{sigma2} = s^2$ ) the inverse Gaussian distribution has density:

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

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

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

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, Whitmore and Seshadri (1987), Aalen and Gjessing (2001) and Gerstein and Mandelbrot (1964).

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

Initial guesses for the inversion of the distribution function used in `qinvgauss` are obtained with the transformation of Whitmore 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>.

Whitmore, 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-Meier
## 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(mu,s2)))
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),col=2)

## End(Not run)

```

**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)
rlllogis(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  $P[X \leq x]$ , 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

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

and density

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

### Value

`dllogis` gives the density, `pllogis` gives the distribution function, `qllogis` gives the quantile function and `rlllogis` 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 <code>length(n) &gt; 1</code> , the length is taken to be the number required.

lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ , 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  $x \geq rp$ .

### 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)
```



**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$neuron=="3",]
## remove unnecessary data
rm(cal2Sdf)
## Extract the elapsed time since the second to last and
## third to last for neuron 1. Normalise the result.
n1.cal2sDF[c("r1N.1","rsN.1","rtN.1")] <- brt4df(n1.cal2sDF,"1N.1",2,c("r1N.1","rsN.1","rtN.1"))
## load mgcv library
library(mgcv)
## fit a model with a tensorial product involving the last
## three spikes and using a cubic spline basis for the last two
## To gain time use a fixed df regression spline
n1S.fitA <- gam(event ~ te(r1N.1,rsN.1,bs="cr",fx=TRUE) + rtN.1,data=n1.cal2sDF,family=binomial(link="logit"))
## transform time
N1.Lambda <- transformedTrain(n1S.fitA)
## check out the resulting spike train using the fact
## that transformedTrain objects inherit from spikeTrain
## objects
N1.Lambda
## Use more formal checks
summary(N1.Lambda)
plot(N1.Lambda,which=c(1,2,4,5),ask=FALSE)
## Transform spike trains of neuron 2 and 3
N2.Lambda <- transformedTrain(n1S.fitA,n2.cal2sDF$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)

```

**Description**

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

**Usage**

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

**Arguments**

lockedTrain	a <a href="#">lockedTrain</a> 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 <a href="#">s</a> .
k	the dimension of the basis used to represent the smooth psth. See <a href="#">s</a> .
x	an <a href="#">gamlockedTrain</a> object.
object	an <a href="#">gamlockedTrain</a> object.
xlim	a numeric (default value supplied). See <a href="#">plot</a> .
ylim	a numeric (default value supplied). See <a href="#">plot</a> .
xlab	a character (default value supplied). See <a href="#">plot</a> .
ylab	a character (default value supplied). See <a href="#">plot</a> .
main	a character (default value supplied). See <a href="#">plot</a> .
lwd	line width used to plot the estimated density. See <a href="#">plot</a> .
col	color used to plot the estimated density. See <a href="#">plot</a> .
...	additional arguments passed to <a href="#">gam</a> in <a href="#">gamlockedTrain</a> . Not used in <a href="#">print.gamlockedTrain</a> and <a href="#">summary.gamlockedTrain</a> . Passed to <a href="#">plot</a> in <a href="#">plot.gamlockedTrain</a> .

**Details**

[gamlockedTrain](#) 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 [gamlockedTrain](#) is returned by [gamlockedTrain](#). This list has the following components:

gamFit	the <a href="#">gamObject</a> generated.
--------	--

Time	the vector of bin centers.
nRef	the number of spikes in the reference train. See <a href="#">hist.lockedTrain</a> .
testFreq	the mean frequency of the test neuron. See <a href="#">hist.lockedTrain</a> .
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.gamlockedTrain` returns the result of `print.gam` applied to the component `gamFit` of its argument.

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

### Author(s)

Christophe Pouzat <[christophe.pouzat@gmail.com](mailto: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 <- gamlockedTrain(lt1.3)
plot(slt1.3)
## do some check on the gam fit
summary(slt1.3)
gam.check(gamObj(slt1.3))
```

---

gammaMLE	<i>Maximum Likelihood Parameter Estimation of a Gamma Model with Possibly Censored Data</i>
----------	---

---

**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 <i>uncensored</i> 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(sh,sc)))
## 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 bootstrap to check the distribution of the deviance
  nbReplicate <- 10000
  sampleSize <- 100
  system.time(
    devianceGA100 <- replicate(nbReplicate,{
      sampGA <- rgamma(sampleSize,shape=shape.true,scale=scale.true)
      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 `gampsth` and `gamlockedTrain` 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 'gampsth'  
gamObj(object, ...)  
## S3 method for class 'gamlockedTrain'  
gamObj(object, ...)
```

**Arguments**

<code>object</code>	an object containing a <code>gamObject</code> . Currently the result of a call to <code>gampsth</code> or to <code>gamlockedTrain</code> .
<code>...</code>	not used for now

**Value**

A `gamObject`

**Author(s)**

Christophe Pouzat <christophe.pouzat@gmail.com>

**See Also**

[gam](#), [gamObject](#), [gampsth](#), [gamlockedTrain](#)

**Examples**

```
##
```

---

gampsth                      *Smooth Peri Stimulus Time Histogram Related Functions and Methods: The Penalized Regression Spline Approach*

---

## Description

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

## Usage

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

## Arguments

<code>repeatedTrain</code>	a <code>repeatedTrain</code> object or a list which can be coerced to such an object.
<code>binSize</code>	the bin size (in s) used to generate the observations on which the gam fit will be performed. See details below.
<code>k</code>	the dimension of the basis used to represent the smooth psth. See <a href="#">s</a> .
<code>bs</code>	the type of splines used. See <a href="#">s</a> .
<code>plot</code>	corresponding argument of <a href="#">hist</a> . Should a plot be generated or not?
<code>object</code>	a <code>gampsth</code> object.
<code>x</code>	a <code>gampsth</code> object.
<code>stimTimeCourse</code>	NULL (default) or a two elements vector specifying the time boundaries (in s) of a stimulus presentation.
<code>colStim</code>	the background color used for the stimulus.
<code>colCI</code>	if not NULL (default) a confidence band is plotted with the specified color; two dashed lines are plotted otherwise.
<code>xlim</code>	a numeric (default value supplied). See <a href="#">plot</a> .
<code>ylim</code>	a numeric (default value supplied). See <a href="#">plot</a> .
<code>xlab</code>	a character (default value supplied). See <a href="#">plot</a> .
<code>ylab</code>	a character (default value supplied). See <a href="#">plot</a> .
<code>main</code>	a character (default value supplied). See <a href="#">plot</a> .

lwd	line width used to plot the estimated density. See <a href="#">plot</a> .
col	color used to plot the estimated density. See <a href="#">plot</a> .
...	in <code>gampsth</code> , if <code>plot</code> is set to <code>TRUE</code> then the ... are passed to <code>plot.gampsth</code> . In <code>plot.gampsth</code> they are passed to <a href="#">plot</a> which is called internally. They are not used otherwise.

### Details

For `gampsth`, 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 `gampsth`, a list of class `gampsth` is returned and no plot is generated. This list has the following components:

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

`print.gampsth` returns the result of [print.gam](#) applied to the `gamObject` generated by `gampsth` and stored in the `environment` of both `lambdaFct` and `LambdaFct`.

`summary.gampsth` returns the result of [summary.gam](#) applied to the `gamObject` generated by `gampsth` and stored in the `environment` of both `lambdaFct` and `LambdaFct`.

### Note

Most of the components of the list returned by `gampsth` 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 gampsth without a plot for neuron 1
## using a cubic regression spline
n1CitrGAMPSTH <- gampsth(e070528citronellal[[1]],plot=FALSE,bs="cr")
## plot the result
plot(n1CitrGAMPSTH,stim=c(6.14,6.64),colCI=2)
## get a summary of the gam fit
summary(n1CitrGAMPSTH)
## perhaps get a more complete check wit gam.check
n1CitrGAMPSTHgo <- gamObj(n1CitrGAMPSTH)
gam.check(n1CitrGAMPSTHgo)
## It does not look too bad
## Now take a look at the observation on which the gam
## was actually performed
plot(n1CitrGAMPSTH$mids,n1CitrGAMPSTH$counts,type="l")
## put dots at the positions of the knots
X <- n1CitrGAMPSTHgo$smooth[[1]][["xp"]]
rug(X,col=2)
## Add the estimated smooth psth after proper scaling
theBS <- diff(n1CitrGAMPSTH[["mids"]])[1]
Y <- n1CitrGAMPSTH$lambdaFct(n1CitrGAMPSTH$mids)*theBS*n1CitrGAMPSTH$nbTrials
lines(n1CitrGAMPSTH$mids,Y,col=4,lwd=2)
```

## Description

Smooths a `lockedTrain` object using a smoothing spline (`gssanova` or `gssanova0`) with the Poisson family after binning the object.

## Usage

```
gsslockedTrain(lockedTrain, bw = 0.001, ...)
gsslockedTrain0(lockedTrain, bw = 0.001, ...)
## S3 method for class 'gsslockedTrain'
print(x, ...)
## S3 method for class 'gsslockedTrain0'
print(x, ...)
## S3 method for class 'gsslockedTrain'
summary(object, ...)
## S3 method for class 'gsslockedTrain0'
summary(object, ...)
## S3 method for class 'gsslockedTrain'
plot(x, xlab, ylab, main, xlim, ylim, col, lwd, ...)
## S3 method for class 'gsslockedTrain0'
plot(x, xlab, ylab, main, xlim, ylim, col, lwd, ...)
```

## Arguments

<code>lockedTrain</code>	a <code>lockedTrain</code> object.
<code>bw</code>	the bin width (in s) used to generate the observations on which the gss fit will be performed. See details below.
<code>x</code>	an <code>gsslockedTrain</code> or a <code>gsslockedTrain0</code> object.
<code>object</code>	an <code>gsslockedTrain</code> or a <code>gsslockedTrain0</code> object.
<code>xlim</code>	a numeric (default value supplied). See <code>plot</code> .
<code>ylim</code>	a numeric (default value supplied). See <code>plot</code> .
<code>xlab</code>	a character (default value supplied). See <code>plot</code> .
<code>ylab</code>	a character (default value supplied). See <code>plot</code> .
<code>main</code>	a character (default value supplied). See <code>plot</code> .
<code>lwd</code>	line width used to plot the estimated density. See <code>plot</code> .
<code>col</code>	color used to plot the estimated density. See <code>plot</code> .
<code>...</code>	in <code>gsslockedTrain</code> , respectively <code>gsslockedTrain0</code> , the ... are passed to the internally called <code>gssanova</code> , respectively <code>gssanova0</code> . Not used in <code>print.gsslockedTrain</code> and <code>summary.gsslockedTrain</code> and their counterparts for <code>gsslockedTrain0</code> objects. Passed to <code>plot</code> in <code>plot.gsslockedTrain</code> and <code>plot.gsslockedTrain0</code> .

## Details

`gsslockedTrain` calls internally `gssanova` while `gsslockedTrain0` calls `gssanova0`. See the respective documentations and references therein for an explanation of the differences. `gsslockedTrain`

and `gsslockedTrain0` essentially generate 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 `gsslockedTrain`, respectively `gsslockedTrain0`, is returned by `gsslockedTrain`, respectively `gsslockedTrain0`. These lists have the following components:

<code>gssFit</code>	the gss object generated by <code>gssanova</code> or <code>gssanova0</code> .
<code>Time</code>	the vector of bin centers.
<code>nRef</code>	the number of spikes in the reference train. See <code>hist.lockedTrain</code> .
<code>testFreq</code>	the mean frequency of the test neuron. See <code>hist.lockedTrain</code> .
<code>bwV</code>	the vector of bin widths used.
<code>CCH</code>	a logical which is TRUE if a cross-intensity was estimated and FALSE in the case of an auto-intensity.
<code>call</code>	the matched call.

`print.gsslockedTrain` returns the result of `print` applied to the `gssanova` object generated by `gsslockedTrain` and stored in the the component `gssFit` of its argument. The same goes for `print.gsslockedTrain0`.

`summary.gsslockedTrain` returns the result of `summary.gssanova` applied to the `gssanova` object generated by `gsspsth` and stored in the component `gssFit` of its argument. The same goes for `summary.gsslockedTrain0`.

### Author(s)

Christophe Pouzat <christophe.pouzat@gmail.com>

### References

Gu C. (2002) *Smoothing Spline ANOVA Models*. Springer.

### See Also

`lockedTrain`, `plot.lockedTrain`, `gssanova`, `gssanova0`

### 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 <- gsslockedTrain(lt1.3)
plot(slt1.3)
## do some check on the gss fit
summary(slt1.3)

## do the same with gsslockedTrain0
slt1.3 <- gsslockedTrain0(lt1.3)
plot(slt1.3)
## do some check on the gss fit
summary(slt1.3)

```

---

gssObj

*Generic Function and Methods for Extracting a gss object*


---

### Description

Some functions of STAR, like `gsspsth`, `gsspsth0` and `gsslockedTrain`, `gsslockedTrain0` perform gss fits internally and keep as a list component or within the environment of a returned function the result of this fit. Method `gssObj` extracts this gss object.

### Usage

```

gssObj(object, ...)
## S3 method for class 'gsspsth'
gssObj(object, ...)
## S3 method for class 'gsspsth0'
gssObj(object, ...)
## S3 method for class 'gsslockedTrain'
gssObj(object, ...)
## S3 method for class 'gsslockedTrain0'
gssObj(object, ...)

```

### Arguments

object	an object containing a <code>gssanova</code> or a <code>gssanova0</code> object. Currently the result of a call to <code>gsspsth</code> , <code>gsspsth0</code> or to <code>gsslockedTrain</code> , <code>gsslockedTrain0</code> .
...	not used for now

### Value

A `gssanova` or a `gssanova0` object.

### Author(s)

Christophe Pouzat <christophe.pouzat@gmail.com>

### See Also

[gssanova](#), [gssanova0](#), [gsspsth](#), [gsspsth0](#), [gsslockedTrain](#), [gsslockedTrain0](#)

**Examples**

```
##
```

---

gsspsth	<i>Smooth Peri Stimulus Time Histogram Related Functions and Methods: The Smoothing Spline Approach</i>
---------	---

---

**Description**

Function `gsspsth` and `gsspsth0` compute a smooth psth, while method `print.gsspsth` and `print.gsspsth0` print and `summary.gsspsth` or `summary.gsspsth0` summarize the `gssanova` / `gssanova0` objects contained in the returned `gsspsth` or `gsspsth0` objects, `plot.gsspsth` or `plot.gsspsth0` plot them and `simulate.gsspsth` or `simulate.gsspsth0` simulate data from fitted objects.

**Usage**

```
gsspsth(repeatedTrain, binSize = 0.025, plot = FALSE, ...)
gsspsth0(repeatedTrain, binSize = 0.025, plot = FALSE, ...)
## S3 method for class 'gsspsth'
print(x, ...)
## S3 method for class 'gsspsth0'
print(x, ...)
## S3 method for class 'gsspsth'
summary(object, ...)
## S3 method for class 'gsspsth0'
summary(object, ...)
## S3 method for class 'gsspsth'
plot(x, stimTimeCourse = NULL, colStim = "grey80",
      colCI = NULL, xlab, ylab, main, xlim, ylim,
      lwd = 2, col = 1, ...)
## S3 method for class 'gsspsth0'
plot(x, stimTimeCourse = NULL, colStim = "grey80",
      colCI = NULL, xlab, ylab, main, xlim, ylim,
      lwd = 2, col = 1, ...)
## S3 method for class 'gsspsth'
simulate(object, nsim = 1, seed = NULL, ...)
## S3 method for class 'gsspsth0'
simulate(object, nsim = 1, seed = NULL, ...)
```

**Arguments**

<code>repeatedTrain</code>	a <code>repeatedTrain</code> object or a list which can be coerced to such an object.
<code>binSize</code>	the bin size (in s) used to generate the observations on which the gss fit will be performed. See details below.
<code>plot</code>	corresponding argument of <code>hist</code> . Should a plot be generated or not?

object	a gsspsth or a gsspsth0 object.
x	a gsspsth or a gsspsth0 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 <a href="#">plot</a> .
ylim	a numeric (default value supplied). See <a href="#">plot</a> .
xlab	a character (default value supplied). See <a href="#">plot</a> .
ylab	a character (default value supplied). See <a href="#">plot</a> .
main	a character (default value supplied). See <a href="#">plot</a> .
lwd	line width used to plot the estimated density. See <a href="#">plot</a> .
col	color used to plot the estimated density. See <a href="#">plot</a> .
nsim	number of repeatedTrain objects to simulate. Defaults to 1.
seed	see <a href="#">simulate</a> .
...	in gsspsth, respectively gsspsth0, the ... are passed to the internally called <a href="#">gssanova</a> , respectively <a href="#">gssanova0</a> . In <a href="#">plot.gsspsth</a> and <a href="#">plot.gsspsth0</a> they are passed to <a href="#">plot</a> which is called internally. They are not used otherwise.

## Details

`gsspsth` calls internally [gssanova](#) while `gsspsth0` calls [gssanova0](#). See the respective documentations and references therein for an explanation of the differences. For both `gsspsth` and `gsspsth0`, 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). Argument `nbasis` of [gssanova](#) called internally by `gsspsth` is set to the number of bins of the histogram resulting from the preprocessing stage.

`simulate.gsspsth` and `simulate.gsspsth0` perform exact simulations of inhomogenous Poisson processes whose (time dependent) rate/intensity function is accessible through the component `lambdaFct` of objects of class `gsspsth` and `gsspsth0`. The inhomogenous Poisson processes are simulated with the thinning method (Devroye, 1986, pp 253-256).

## Value

When `plot` is set to `FALSE` in `gsspsth`, respectively `gsspsth0`, a list of class `gsspsth`, respectively `gsspsth0`, is returned and no plot is generated. These list have 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 <a href="#">predict.ssanova</a> for details.

ciLow	a vector with the lower limit of a pointwise 95% confidence interval.
breaks	a vector with 2 elements the earliest 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. <a href="#">integrate</a> 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.gsspsth` with a `gsspsth` object argument or `plot.gsspsth0` with a `gsspsth0`.

`print.gsspsth` returns the result of `print` applied to the `gssanova` object generated by `gsspsth` and stored in the `environment` of both `lambdaFct` and `LambdaFct`. The same goes for `print.gsspsth0`.

`summary.gsspsth` returns the result of `summary.gssanova` applied to the `gssanova` object generated by `gsspsth` and stored in the `environment` of both `lambdaFct` and `LambdaFct`. The same goes for `summary.gsspsth0`.

`simulate.gsspsth` and `simulate.gsspsth0` return a `repeatedTrain` object if argument `nsim` is set to one and a list of such objects if it is greater than one.

### Note

Most of the components of the list returned by `gsspsth` and `gsspsth0` 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](mailto:christophe.pouzat@gmail.com)>

### References

Gu C. (2002) *Smoothing Spline ANOVA Models*. Springer.

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.

Devroye Luc (1986) *Non-Uniform Random Variate Generation*. Springer. Book available in pdf format at: <http://cg.scs.carleton.ca/~luc/rnbookindex.html>.

### See Also

[psth](#), [plot.psth](#), [gssanova](#), [gssanova0](#), [summary.gssanova](#), [summary.gssanova0](#), [reportHTML.repeatedTrain](#), [simulate](#)

## Examples

```

## Get the e070528citronellal data set into workspace
data(e070528citronellal)
## Compute gsspsth without a plot for neuron 1
## using a smoothing spline with gssanova0, and default bin size of 25 ms.
n1CitrGSSPSTH0 <- gsspsth0(e070528citronellal[[1]])
## plot the result
plot(n1CitrGSSPSTH0,stim=c(6.14,6.64),colCI=2)
## get a summary of the gss fit
summary(n1CitrGSSPSTH0)
## Now take a look at the observation on which the gss
## was actually performed
plot(n1CitrGSSPSTH0$mids,n1CitrGSSPSTH0$counts,type="l")
## Add the estimated smooth psth after proper scaling
theBS <- diff(n1CitrGSSPSTH0[["mids"]])[1]
Y <- n1CitrGSSPSTH0$lambdaFct(n1CitrGSSPSTH0$mids)*theBS*n1CitrGSSPSTH0$nbTrials
lines(n1CitrGSSPSTH0$mids,Y,col=4,lwd=2)
## Not run:
## check the (absence of) effect of the pre-binning by using a smaller
## and larger one, say 10 and 75 ms
n1CitrGSSPSTH0_10 <- gsspsth0(e070528citronellal[[1]],binSize=0.01)
n1CitrGSSPSTH0_75 <- gsspsth0(e070528citronellal[[1]],binSize=0.075)
## plot the "high resolution" smoothed-psth
plot(n1CitrGSSPSTH0_10,colCI="grey50")
## add to it the estimate obtained with the "low resolution" one
Y_75 <- n1CitrGSSPSTH0_75$lambdaFct(n1CitrGSSPSTH0_10$mids)
lines(n1CitrGSSPSTH0_10$mids,Y_75,col=2,lwd=2)

## End(Not run)
## simulate data from the first fitted model
s1 <- simulate(n1CitrGSSPSTH0)
## look at the result
s1

## Not run:
## Do the same thing with gsspsth
n1CitrGSSPSTH <- gsspsth(e070528citronellal[[1]])
plot(n1CitrGSSPSTH,stim=c(6.14,6.64),colCI=2)
summary(n1CitrGSSPSTH)
plot(n1CitrGSSPSTH$mids,n1CitrGSSPSTH$counts,type="l")
theBS <- diff(n1CitrGSSPSTH[["mids"]])[1]
Y <- n1CitrGSSPSTH$lambdaFct(n1CitrGSSPSTH$mids)*theBS*n1CitrGSSPSTH$nbTrials
lines(n1CitrGSSPSTH$mids,Y,col=4,lwd=2)
## check the (absence of) effect of the pre-binning by using a smaller
## and larger one, say 10 and 75 ms
n1CitrGSSPSTH_10 <- gsspsth(e070528citronellal[[1]],binSize=0.01)
n1CitrGSSPSTH_75 <- gsspsth(e070528citronellal[[1]],binSize=0.075)
## plot the "high resolution" smoothed-psth
plot(n1CitrGSSPSTH_10,colCI="grey50")
## add to it the estimate obtained with the "low resolution" one
Y_75 <- n1CitrGSSPSTH_75$lambdaFct(n1CitrGSSPSTH_10$mids)
lines(n1CitrGSSPSTH_10$mids,Y_75,col=2,lwd=2)

```

```
## simulate data from the first fitted model
s1 <- simulate(n1CitrGSSPSTH)
## look at the result
s1

## End(Not run)
```

---

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)
hrexpl(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 refractory exponential.  
 log                    should the log hazard be returned? FALSE by default.

### Details

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

### 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 <a href="#">lockedTrain</a> 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 <a href="#">plot</a> . Default supplied.
ylim	a numeric. See <a href="#">plot</a> . Default supplied.
type	see <a href="#">lines</a> . Default supplied.
pch	see <a href="#">plot</a> . Default supplied.
...	see <a href="#">plot</a> .

## Details

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

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

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):

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

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:

<code>density</code>	the density estimate. Equivalent of the component density returned by <code>hist</code> .
<code>breaks</code>	a numeric vector with the breaks in between which spikes were counted. Similar to the component of the same name returned by <code>hist</code> .
<code>mids</code>	a numeric vector with the mid points of breaks. . Similar to the component of the same name returned by <code>hist</code> .
<code>bw</code>	the bin width used.
<code>nRef</code>	the total number of reference spikes used.
<code>refFreq</code>	the mean frequency of the reference neuron.
<code>testFreq</code>	the mean frequency of the test neuron.
<code>obsTime</code>	the total observation time used (in s).
<code>CCH</code>	a logical which is <code>TRUE</code> if a cross-intensity was estimated and <code>FALSE</code> in the case of an auto-intensity.
<code>call</code>	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, Yoshihiko (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$Date),,c(0,500))
shalShocksH <- hist(shalShocks,5,plot=FALSE)
plot(shalShocksH,"Ogata",c(0.95,0.99),xlab="TIME LAG (DAYS)",ylab="NUMBER OF EVENTS PER DAY")
## Reproduce Fig. 7 of Ogata 1988
myBinNb <- 101
myMidPoints <- seq(from = 1, to = 6, length.out=myBinNb)
myMidPoints <- 10^myMidPoints/200
myBreaks <- c(0,myMidPoints[-myBinNb] + diff(myMidPoints) / 2)
shalShocksH2 <- hist(shalShocks,breaks=myBreaks,plot=FALSE)
yy <- abs(shalShocksH2$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 <i>uncensored</i> 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:

<code>estimate</code>	the estimated parameters, a named vector.
<code>se</code>	the standard errors, a named vector.
<code>logLik</code>	the log likelihood at maximum.
<code>r</code>	a function returning the log of the relative likelihood function.
<code>mll</code>	a function returning the opposite of the log likelihood function using the log of the parameters.
<code>call</code>	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 therefore
## 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:
##  $I(x,y) \leftarrow t(Q) \%*\% I(u,v) \%*\% Q$ 
## 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)
sampIGmleIG$contour <- sapply(Mu, function(mu) sapply(Sigma2, function(s2) sampIGmleIG$r(mu,s2)))
## 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(sampIGmleIG$contour),
        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] - qchisq(CI,1)/2

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] - qchisq(CI,1)/2

## 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 bootstrap to check the distribution of the deviance
nbReplicate <- 1000 #10000
sampleSize <- 100
system.time(
devianceIG100 <- 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$mll(c(logMu,x))+logLik(sampIGmleIG),...)$objective
}
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$mll(c(x,logS2))+logLik(sampIGmleIG),...)$objective
}
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[1]),
  CI=0.95)$root,
  uniroot(logMuProfCI,
  interval=c(logPara[1],intervalMu(factor)[2]),
  CI=0.95)$root
)
logMuCI99 <- c(uniroot(logMuProfCI,
  interval=c(intervalMu(factor)[1],logPara[1]),
  CI=0.99)$root,
  uniroot(logMuProfCI,
  interval=c(logPara[1],intervalMu(factor)[2]),
  CI=0.99)$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[2]),
  CI=0.95)$root,
  uniroot(logS2ProfCI,
  interval=c(logPara[2],intervalS2(factor)[2]),
  CI=0.95)$root
)
logS2CI99 <- c(uniroot(logS2ProfCI,
  interval=c(intervalS2(factor)[1],logPara[2]),
  CI=0.99)$root,
  uniroot(logS2ProfCI,
  interval=c(logPara[2],intervalS2(factor)[2]),
  CI=0.99)$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)

```

---

 isi

*Get Lagged Inter Spike Intervals (ISIs) From Data Frames Generated  
by mkGLMdf*

---

### Description

A utility function to create a vector containing the *i*th preceding inter spike interval (isi) at a given time.

### Usage

```
isi(dataFrame, lag = 1)
```

**Arguments**

dataFrame	a <a href="#">data.frame</a> typically generated by <a href="#">mkGLMdf</a> . Should at least contain an event and a time variable.
lag	a strictly positive integer. Set to 1 if the previous isi is required, to 2 is the isi preceding the last one is required, etc...

**Details**

Look at the (short) source file for details.

**Value**

A numeric vector with the value of the lagth isi preceding the time of the corresponding bin center.

**Note**

Before plugging the result into [gssanova](#), do not forget to remove the NA elements (see the example).

**Author(s)**

Christophe Pouzat <[christophe.pouzat@gmail.com](mailto:christophe.pouzat@gmail.com)>

**See Also**

[mkGLMdf](#), [gssanova](#), [%tt%](#)

**Examples**

```
## Not run:
## load e060517spont data set
data(e060517spont)
## make a data frame using a 2 ms bin width
e060517spontDF <- mkGLMdf(e060517spont,0.002,0,60)
## Keep data relevant to neuron 1
e060517spontDFn1 <- e060517spontDF[e060517spontDF$neuron == "1",]
## get the isi at lag 1 and 2
e060517spontDFn1$isi1 <- isi(e060517spontDFn1,lag=1)
e060517spontDFn1$isi2 <- isi(e060517spontDFn1,lag=2)
## keep only defined elements
e060517spontDFn1 <- e060517spontDFn1[!is.na(e060517spontDFn1$isi2),]
## split the data set into an "early" and a "late" part
e060517spontDFn1e <- e060517spontDFn1[e060517spontDFn1$time <= 30,]
e060517spontDFn1l <- e060517spontDFn1[e060517spontDFn1$time > 30,]
## Fit the late part
e060517spontDFn1lGF <- gssanova(event ~ 1N.1*isi1+isi2, data=e060517spontDFn1l, family="binomial", seed=20061001)
## Time transform the early part and perform goodness of fit tests
e060517spont.n1e.tt <- e060517spontDFn1lGF %tt% e060517spontDFn1e
e060517spont.n1e.tt
summary(e060517spont.n1e.tt)
plot(summary(e060517spont.n1e.tt))
```

```
## 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 <a href="#">hist</a> .

### 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 beginning and the end of the acquisition. If missing values are obtained using the <a href="#">floor</a> of the smallest spike time and the <a href="#">ceiling</a> of the largest one.
nbEvtPerBin	If both xBreaks and yBreaks 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 <a href="#">plot</a> .
ylab	a character (default value supplied). See <a href="#">plot</a> .
main	a character (default value supplied). See <a href="#">plot</a> .
pch	the type of "points" displayed by jsd. See <a href="#">plot</a> .
...	additional arguments passed to <a href="#">plot</a> by jsd and to respective generic methods by <code>contour.jpsth</code> , <code>image.jpsth</code> and <code>persp.jpsth</code> .

**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 reformatting of a `jpsth` object in order to compute a smooth version of it with [gssanova](#), [gssanova0](#) or [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 frequency of the "X" neuron.
yFreq	the mean frequency 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 peris-timulus 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](#), [gsslockedTrain](#), [plot.gsslockedTrain](#), [gsslockedTrain0](#), [plot.gsslockedTrain0](#), [gamlockedTrain](#), [plot.gamlockedTrain](#), [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)
## Not run:
## fit a gss model with interactions
```

```

## use a larger bin width for the jpsth
j1.2 <- jpsth(e070528citronellal[[1]],e070528citronellal[[2]],0.2,0.2)
## get a data frame
j1.2DF <- jpsth2df(j1.2)
## To save computation time start analyzing
## just before the stimulation time
j1.2DF <- j1.2DF[j1.2DF$X > 6 & j1.2DF$Y>6,]
gf <- gssanova(Count ~ X*Y, family="poisson", data=j1.2DF,seed=20061001)
## Use the project function of gss to check the significance
## of the interaction term
project(gf2,inc=c("X","Y"))

## End(Not run)
## Not run:
## fit a gam model assuming no interaction
## get a data frame
j1.2DF <- jpsth2df(j1.2)
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 <i>uncensored</i> observations for each value of yi; default: numeric(length(yi))+1.

## Details

The MLE for the log logistic is not available in closed form 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:

<code>estimate</code>	the estimated parameters, a named vector.
<code>se</code>	the standard errors, a named vector.
<code>logLik</code>	the log likelihood at maximum.
<code>r</code>	a function returning the log of the relative likelihood function.
<code>mll</code>	a function returning the opposite of the log likelihood function using the log of parameter <code>sdlog</code> .
<code>call</code>	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 logistic
## distribution
set.seed(1102006, "Mersenne-Twister")
sampleSize <- 100
location.true <- -2.7
scale.true <- 0.025
sampLL <- rlogis(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 bootstrap to check the distribution of the deviance
nbReplicate <- 10000
sampleSize <- 100
system.time(
  devianceLL100 <- replicate(nbReplicate,{
    sampLL <- rllgis(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)

```

---

InormMLE

*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

```

InormMLE(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:

<code>estimate</code>	the estimated parameters, a named vector.
<code>se</code>	the standard errors, a named vector.
<code>logLik</code>	the log likelihood at maximum.
<code>r</code>	a function returning the log of the relative likelihood function.
<code>mll</code>	a function returning the opposite of the log likelihood function using the log of parameter <code>sdlog</code> .
<code>call</code>	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 <- InormMLE(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 InormMLE 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 therefore
## 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:
##  $I(x,y) \leftarrow t(Q) \%*\% I(u,v) \%*\% Q$ 
## 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$r(mu,s)))
## plot contours using a linear scale for the parameters
## draw four contours corresponding to the following likelihood ratios:
## 0.5, 0.1, Chi2 with 2 df and p values of 0.95 and 0.99
X11(width=12,height=6)
layout(matrix(1:2,ncol=2))
contour(MeanLog,SdLog,t(samplNmleLNcontour),
       levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
       labels=c("log(0.5)",
               "log(0.1)",
               "-1/2*P(Chi2=0.95)",
               "-1/2*P(Chi2=0.99)"),
       xlab=expression(mu),ylab=expression(sigma),
       main="Log Relative Likelihood Contours"
       )
points(samplNmleLN$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 bootstrap to check the distribution of the deviance
nbReplicate <- 10000
sampleSize <- 100
system.time(
    devianceLN100 <- replicate(nbReplicate,{
        sampLN <- rlnorm(sampleSize,meanlog=meanlog.true,sdlog=sdlog.true)
        sampLNmleLN <- lnormMLE(sampLN)
        -2*sampLNmleLN$r(meanlog.true,sdlog.true)
    })
)

## 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)

```

**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 beginning and the end of the acquisition. If missing values are obtained using the <a href="#">floor</a> of the smallest spike time and the <a href="#">ceiling</a> 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 <a href="#">plot</a> .
pch	data symbol used for the spikes. See <a href="#">plot</a> .
xlab	a character (default value supplied). See <a href="#">plot</a> .
ylab	a character (default value supplied). See <a href="#">plot</a> .
main	a character (default value supplied). See <a href="#">plot</a> .
...	see <a href="#">plot</a> or <a href="#">print</a> .

### 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"]]
## construct 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)
```

## Description

The variables added to the data frame corresponding to the first argument of the function are the former inter spike intervals. These variables are moreover transformed with mkM2U so that they have an approximately uniform distribution on their definition domain.

**Usage**

```
mkAR(df, low, high, max.order, selfName = "IN.1",...)
```

**Arguments**

df	a data frame. This data frame should contain a variable <code>time</code> like data frames returned by <code>mkGLMdf</code> .
low	a numeric, the smallest value of variable <code>time</code> from which the transformation is looked for. If missing defaults to the smallest time.
high	a numeric, the largest value of variable <code>time</code> up to which the transformation is looked for. If missing defaults to the largest time.
max.order	a positive integer, the maximal order of the AR model. How many previous inter spike intervals should be used in order to predict the duration of the next interval?
selfName	a character string or an integer specifying the variable of <code>df</code> containing the elapsed time since the last spike of the considered neuron.
...	additional arguments passed to <code>mkM2U</code> .

**Details**

When `max.order > 1` the previous inter spike intervals are all transformed using the "map to uniform" function estimated from the inter spike intervals at lag 1.

**Value**

A data frame is returned. In addition to the variables of `df` the returned data frame contains a variable `est` with the transformed elapsed time since the last spike of the neuron and `i1t, i2t, ..., i` `max.order - t`, the transformed previous inter spike intervals. The returned data frame has also four attributes:

<code>fm1a</code>	a formula suitable for a first argument of, say, <code>gssanova</code> .
<code>m2uL</code>	the function returned by <code>mkM2U</code> transforming the elapsed time since the last spike of the neuron.
<code>m2uI</code>	the function returned by <code>mkM2U</code> transforming the first former inter spike interval.
<code>call</code>	the matched call.

**Author(s)**

Christophe Pouzat <[christophe.pouzat@gmail.com](mailto:christophe.pouzat@gmail.com)>

**See Also**

[mkM2U](#), [gssanova](#)

**Examples**

```
## Not run:
require(STAR)
data(e060824spont)
DFA <- subset(mkGLMdf(e060824spont,0.004,0,59),neuron==1)
DFA <- mkAR(DFA, 0, 29, 5, maxiter=200)
head(DFA)
tail(DFA)
ar.fit <- gssanova(attr(DFA,"fmla"), data=DFA,family="binomial",seed=20061001)
plot(ar.fit %qp% "est")
plot(ar.fit %qp% "i1t")
plot(ar.fit %qp% "i2t")
plot(ar.fit %qp% "i3t")
plot(ar.fit %qp% "i4t")
plot(ar.fit %qp% "i5t")

## End(Not run)
```

mkCPSP

*Counting Process Sample Paths***Description**

Functions to create and explore CountingProcessSamplePath objects. These objects are complementary to the spikeTrain objects, the latter being in fact point processes representations.

**Usage**

```
mkCPSP(st, from = floor(min(st)), to = ceiling(max(st)))
as.CPSP(x)
## S3 method for class 'CountingProcessSamplePath'
print(x, digits = 5, ...)
## S3 method for class 'CountingProcessSamplePath'
plot(x, y, col, lwd, xlim, ylim,
      xlab, ylab, xaxs, yaxs, main, ...)
## S3 method for class 'CountingProcessSamplePath'
lines(x, ...)
```

**Arguments**

st	A numeric vector with <i>strictly</i> increasing elements.
from	A numeric, the time at which the counting process observation started.
to	A numeric, the time at which the counting process observation ended.
x	A numeric or a spikeTrain object for as.CPSP, a CountingProcessSamplePath object for print, plot and lines.
digits	An integer, the number of digits to be used while printing summaries. See <a href="#">round</a> .

`y` Not used but required by the `plot` method definition.  
`col, lwd, xlim, ylim, xlab, ylab, main, xaxs, yaxs`  
 See [plot](#).  
`...` Not used in `print` (but included for compatibility with the method definition)  
 otherwise used like in [plot](#) and [lines](#).

### Details

CountingProcessSamplePath objects are complementary to spikeTrain objects. They are also used to represent slightly more general properties of these objects and are directed towards model testing.

More formally, if we observe  $n$  events at times  $\{t_1, \dots, t_n\}$  such that,  $from < t_1 < \dots < t_n \leq to$ , the *counting process sample path* is the right continuous function defined by:

$$N(t) = \#\{t_j \text{ with } from < t_j \leq t\}$$

where  $\#$  stands for the number of elements of a set.

### Value

mkCPSP returns an object of class CountingProcessSamplePath. This object is a list with the following components:

<code>cspFct</code>	a right continuous function of <code>t</code> returning the number of events whose occurrence time is strictly larger than <code>from</code> and smaller of equal than <code>t</code> . <code>t</code> can be a vector. If missing the cumulative number of events at the events occurrence time is returned.
<code>ppspFct</code>	a function that does not take any argument and that returns the sequence of events times, that is, the "point process sample path".
<code>spikeTrainFct</code>	a function that does not take any argument and that returns the spikeTrain object associated with the CountingProcessSamplePath object.
<code>from</code>	argument <code>from</code> of <code>mkCPSP</code> .
<code>to</code>	argument <code>to</code> of <code>mkCPSP</code>
<code>call</code>	the matched call.

Functions `plot` and `lines` are used for their side effects, function `print` returns a short description of the object corresponding to the summary returned by function [summary.spikeTrain](#) for spikeTrain objects. Function `as.CPSP` returns a CountingProcessSamplePath.

### Note

This functions are directed towards model testing, don't be surprised if they look redundant with the corresponding functions for spikeTrain objects. An apparent difference of detail with the latter is that no scale (like seconds) is assumed by default for CountingProcessSamplePath objects. This is to cope in a natural way with the time transformation / rescaling procedures used to test conditional intensity models.

**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**

[summary.CountingProcessSamplePath](#), [print.CountingProcessSamplePath.summary](#), [plot.CountingProcessSamplePath.summary](#), [summary.spikeTrain](#), [print.spikeTrain](#), [plot.spikeTrain](#), [as.spikeTrain](#)

**Examples**

```
## A simple illustration with Ogata's hearthquakes data set
data(ShallowShocks)
plot(mkCPSP(ShallowShocks$Date),
     xlab="Time (days)",
     main="Shallow Shocks Counting Process of Ogata 1988")
## An illustration with on of STAR's data neuroanl discharge data set
data(e060824spont)
## Create the object from a spikeTrain
n1spt.cp <- as.CPSP(e060824spont[["neuron 1"]])
## print it
n1spt.cp
## plot it
plot(n1spt.cp)
```

---

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`, `gss` and `gam` to be used with the `poisson` or `binomial` family to fit the spike trains.

**Usage**

```
mkGLMdf(obj, delta, lwr, upr)
```

**Arguments**

<code>obj</code>	a <code>spikeTrain</code> or a <code>repeatedTrain</code> objects or a list of any of those two.
<code>delta</code>	the bin size used for time discretization (in s).
<code>lwr</code>	the time (in s) at which the recording window starts. If missing a value is obtained using the <code>floor</code> of the smallest spike time.
<code>upr</code>	the time (in s) at which the recording window ends. If missing a value is obtained using the <code>ceiling</code> of the largest spike time.

**Details**

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.
IN.x	a numeric. 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, Val'erie (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](#), [gssanova](#), [mgcv](#), [as.spikeTrain](#), [as.repeatedTrain](#)

**Examples**

```
## Analysis of a "simple" spontaneous train
## load the data
data(e060824spont)
## create a data frame using the 1st neuron
DFA <- subset(mkGLMdf(e060824spont,0.004,0,59),neuron==1)
## Add the previous ISI to the data frame
DFA <- within(DFA,i1 <- isi(DFA,lag=1))
DFA <- DFA[complete.cases(DFA),]
## estimate the "map to uniform" functions for 2 variables:
```

```

## 1) the elapsed time since the last spike (lN.1)
## 2) the previous insterspike interval
## Do this estimation on the first half of the set
m2u1 <- mkM2U(DFA,"lN.1",0,29)
m2ui <- mkM2U(DFA,"i1",0,29,maxiter=200)
## create "mapped" variables
DFA <- within(DFA,e1t <- m2u1(lN.1))
DFA <- within(DFA,i1t <- m2ui(i1))
## split the data in 2 parts, one for the fit, the other for the test
DFAe <- subset(DFA, time <= 29)
DFA1 <- subset(DFA, time > 29)
## fit an additive model with gssanova
m1.fit <- gssanova(event ~ e1t + i1t,
                  data = DFAe,
                  family="binomial",
                  seed=20061001)
## test the model by "time transforming" the late part
tt.l <- m1.fit %tt% DFA1
tt.l.summary <- summary(tt.l)
tt.l.summary
plot(tt.l.summary,which=c(1,2,4,6))

## 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 presynaptic 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 function, 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),c(0,60))
interData.lt2 <- lockedTrain(interData[["pre"]],interData[["post2"]],laglim=c(-0.03,0.05),c(0,60))
interData.lt3 <- lockedTrain(interData[["pre"]],interData[["post3"]],laglim=c(-0.03,0.05),c(0,60))
## 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(-0.03,0.05),c(0,60))
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$neuron=="2",]
## fit dfN2 with a smooth effect for the elapsed time since the last
## event of neuron 2 and another one with the elapsed time since the
## last event from neuron 1. Use moreover only the events for which the
## the last event from neuron 1 occurred at most 100 ms ago.
dfN2.fit0 <- gam(event ~ s(lN.1,bs="cr") + s(lN.2,bs="cr"), data=dfN2, family=poisson, subset=(dfN2$lN.1 <=0.1))
## 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 presynaptic 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)*0.001,lN.2=(1:300)*0.001))),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)*0.001,lN.2=(1:300)*0.001))))) ,type="l")
lines(xx,plnorm(xx,-2.4,0.4,lower.tail=FALSE),col=2)

## use gssanova
## split the data set in 2 parts, one for the fit, the other for the
## test
dfN2e <- dfN2[dfN2$time <= 20,]
dfN2l <- dfN2[dfN2$time > 20,]
## fit the same model as before with gssanova
dfN2.fit1 <- gssanova(event ~ lN.1 + lN.2, data=dfN2e, family="poisson", seed=20061001)
## plot the effect of neuron 1
pred1 <- predict(dfN2.fit1,data.frame(lN.1=seq(0.001,0.220,0.001),
                                     lN.2=rep(median(dfN2e$lN.2),220)),
                se=TRUE)
plot(seq(0.001,0.220,0.001),
     pred1$fit,type="l",
     ylim=c(min(pred1$fit-1.96*pred1$se.fit),max(pred1$fit+1.96*pred1$se.fit))
)
lines(seq(0.001,0.220,0.001),pred1$fit-1.96*pred1$se.fit,lty=2)
lines(seq(0.001,0.220,0.001),pred1$fit+1.96*pred1$se.fit,lty=2)
## transform the time of the late part of the train
## first make sure than lN.1 and lN.2 are within the right bounds
m1 <- max(dfN2e$lN.1)
m2 <- max(dfN2e$lN.2)
dfN2l$lN.1 <- sapply(dfN2l$lN.1, function(x) min(m1,x))
dfN2l$lN.2 <- sapply(dfN2l$lN.2, function(x) min(m2,x))
pred1 <- predict(dfN2.fit1,dfN2l)
Lambda <- cumsum(exp(pred1))
ttl <- mkCPSP(Lambda[dfN2l$event==1])
ttl

```

```

plot(summary(ttl))
## see what happens without time transformation
rtl <- mkCPSP(dfN2$time[dfN2$event==1])
plot(summary(rtl))

## 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, family=poisson, subset=(
## 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$neuron=="3",]
dfN3.fit0 <- gam(event ~ s(lN.1,k=20,bs="cr") + s(lN.3,k=15,bs="cr"),data=dfN3,family=poisson, subset=(dfN3$lN.1
summary(dfN3.fit0)
plot(dfN3.fit0,select=1,ylim=c(-1.5,1.8),rug=FALSE)
dfN4 <- dfAll[dfAll$neuron=="4",]
dfN4.fit0 <- gam(event ~ s(lN.1,k=20,bs="cr") + s(lN.4,k=15,bs="cr"),data=dfN4,family=poisson, subset=(dfN4$lN.1
summary(dfN4.fit0)
plot(dfN4.fit0,select=1,ylim=c(-1.8,1.5),rug=FALSE)

## End(Not run)

```

mkM2U

*Makes a Smooth Function Mapping a Data Frame Variable Onto a Variable Uniform on Its Definition Domain*

## Description

The smooth transformation function is a smooth version of the [ecdf](#). A smooth density estimate as well as the inverse transformation (the quantile function) are also returned as attributes.

## Usage

```
mkM2U(df, vN, low, high, delta, alpha=2, ...)
```

## Arguments

df	a data frame. This data frame should contain a variable time like data frames returned by <a href="#">mkGLMdf</a> .
vN	a character string corresponding to the name of one of the variables of df or an integer, its index. Variable vN is the one for which the mapping to uniform is looked for.
low	a numeric, the smallest value of variable time from which the transformation is looked for. If missing defaults to the smallest time.
high	a numeric, the largest value of variable time up to which the transformation is looked for. If missing defaults to the largest time.
delta	a numeric, the bin width used to build the variable values histogram. This histogram is subsequently smoothed. Default provided if missing.

alpha            see [ssden](#).  
 ...              additional arguments passed to [ssden](#) called internally by the function

### Details

The smooth mapping to uniform function returned by `mkM2U` is obtained by first selecting a subset of the variable values for which the variable `time` of `df` is between `low` and `high`. The values are then binned between the `min` and the `max` of the (complete) variable values with a bin width `delta`. Function [ssden](#) is then called on the histogram and the result is stored in object `ii.fit` (This object is stored in the closure of the returned function). The returned function is the result of a call of `pssden` on `ii.fit` and the argument.

A function inverting the "mapping to uniform function", that is, a quantile function, is also returned as `attributes` `qFct`. This inverse function is obtained by numerical inversion, calling [uniroot](#) internally. Additional arguments can be passed to [uniroot](#) via the `...` argument of the function.

A function returning the smooth density estimate is returned as `attributes` `dFct`.

### Value

A function returning the probability for  $vN$  random variable to have a value smaller or equal to its first argument. The returned function calls internally [integrate](#). Additional arguments can be passed to the latter via the `...` argument of the returned function.

As explained in the `details` section, the returned function has the smooth density function, `dFct`, as well as the inverse function, `qFct`, as `attributes`. Attribute `call` contains the matched call and `range` contains the full range of the mapped variable.

### Note

Since the density returned by [dssden](#) can sometime integrate to a value slightly different from 1 on its definition domain, the actual integral is evaluated with [integrate](#) and the returned density is renormalised. A look-up table of 101 regularly spaced quantiles and the corresponding probabilities is also created and stored in the returned function closure. This look-up table is used to speed up the computations performed by the returned function which uses [integrate](#) and not `pssden`. It is also used to speed up the computations of the inverse function (returned as attribute `qFct`) which uses [uniroot](#) and not `qssden`.

### Author(s)

Christophe Pouzat <[christophe.pouzat@gmail.com](mailto:christophe.pouzat@gmail.com)>

### See Also

[ssden](#), [dssden](#), [integrate](#), [uniroot](#), [mkGLMdf](#)

### Examples

```
require(STAR)
data(e060824spont)
DFA <- subset(mkGLMdf(e060824spont, 0.004, 0, 59), neuron==1)
DFA <- within(DFA, i1 <- isi(DFA, lag=1))
```

```

DFA <- DFA[complete.cases(DFA),]
m2u1 <- mkM2U(DFA,"1N.1",0,29)
m2ui <- mkM2U(DFA,"i1",0,29,maxiter=200)
DFA <- within(DFA,e1t <- m2u1(1N.1))
DFA <- within(DFA,i1t <- m2ui(i1))
with(DFA,plot(ecdf(e1t[time>29]),pch="."))
abline(a=0,b=1,col=2,lty=2)
with(DFA,plot(ecdf(i1t[time>29]),pch="."))
abline(a=0,b=1,col=2,lty=2)

```

---

mkREdf	<i>Evaluates RateEvolutions for spikeTrain Lists and Returns Data Frame</i>
--------	---

---

## 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 <i>named</i> 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 <a href="#">rateEvolution</a> . This can be a vector.
kernel	see <a href="#">rateEvolution</a> .
n	see <a href="#">rateEvolution</a> .
from	see <a href="#">rateEvolution</a> .
to	see <a href="#">rateEvolution</a> .
na.rm	see <a href="#">rateEvolution</a> .
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, below it
ask	logical; if TRUE, the user is <i>asked</i> before each plot, see <code>par(ask=.)</code> .
...	additional arguments passed to <code>plot</code> .

**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$neuron=="3",]
## remove unnecessary data
rm(cal2Sdf)
## Extract the elapsed time since the second to last and
## third to last for neuron 1. Normalise the result.
n1.cal2sDF[c("r1N.1","rsN.1","rtN.1")] <- brt4df(n1.cal2sDF,"1N.1",2,c("r1N.1","rsN.1","rtN.1"))
## load mgcv library
library(mgcv)
## fit a model with a tensorial product involving the last
## three spikes and using a cubic spline basis for the last two
## To gain time use a fixed df regression spline
n1S.fitA <- gam(event ~ te(r1N.1,rsN.1,bs="cr",fx=TRUE) + rtN.1,data=n1.cal2sDF,family=binomial(link="logit"))
## transform time
N1.Lambda <- transformedTrain(n1S.fitA)
## check out the resulting spike train using the fact
## that transformedTrain objects inherit from spikeTrain
## objects
N1.Lambda
## Use more formal checks
summary(N1.Lambda)
plot(N1.Lambda,which=c(1,2,4,5),ask=FALSE)
## Transform spike trains of neuron 2 and 3
N2.Lambda <- transformedTrain(n1S.fitA,n2.cal2sDF$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.quickPredict

*Graphical Methods for quickPredict Objects*


---

## Description

plot, lines, image, contour and persp methods for [quickPredict](#) objects.

## Usage

```
## S3 method for class 'quickPredict'
```

```

plot(x, y, xFct = function(x) x, style = c("band",
      "simple"), ylim, meanCol = 2, meanWD = 2, bandCol = "grey50", xlab,
      ylab, ...)
## S3 method for class 'quickPredict'
lines(x, what = c("mean", "upr", "lwr"),
      xFct = function(x) x, ...)
## S3 method for class 'quickPredict'
image(x, main, xlab, ylab, ...)
## S3 method for class 'quickPredict'
contour(x, what = c("mean", "sd"), main, xlab, ylab, add,
      ...)
## S3 method for class 'quickPredict'
persp(x, what = c("mean", "sd"), main, xlab, ylab, zlab, ...)

```

### Arguments

x	a <a href="#">quickPredict</a> object.
y	Not used but required by the plot method definition.
xFct	a function applied to the xx component of argument x which is itself a <a href="#">quickPredict</a> object. Useful to get a plot with the "native" scale of the variable.
style	a character string, either "band" or "simple". Controls the way confidence intervals are displayed. If "band" is selected they are shown as a ribbon. Their boundaries appear as dashed lines otherwise.
ylim	see <a href="#">plot</a> (default provided if missing).
meanCol	the color used to display the estimated mean of the term (see argument col in <a href="#">plot</a> ).
meanWD	the width used to display the estimated mean of the term (see argument lwd in <a href="#">plot</a> ).
bandCol	the color of the confidence interval ribbon when one is is drawn.
xlab, ylab, zlab	see <a href="#">plot</a> and <a href="#">persp</a> (default provided if any is missing).
main	see <a href="#">image</a> , <a href="#">contour</a> and <a href="#">persp</a> (default provided if missing).
what	for <a href="#">lines.quickPredict</a> , one of the following character strings: "mean", "upr", "lwr". Controls the line drawn: the estimated mean, upper bound of the 95% CI or lower bound. For <a href="#">contour.quickPredict</a> and <a href="#">persp.quickPredict</a> , a character string specifying if the mean or the sd contours or surface should be drawn.
add	see <a href="#">contour</a> .
...	additional arguments to <a href="#">plot</a> , <a href="#">lines</a> , <a href="#">image</a> , <a href="#">contour</a> and <a href="#">persp</a> .

### Value

Nothing returned, these functions are used for their side effects: plots are generated or modified.

### Author(s)

Christophe Pouzat <[christophe.pouzat@gmail.com](mailto:christophe.pouzat@gmail.com)>

**See Also**

[quickPredict](#), [plot](#), [lines](#) [image](#), [contour](#), [persp](#), [plot.ssanova](#)

**Examples**

```
## Not run:
## Follow up of ssanova example of gss
data(nox)
nox.fit <- ssanova(log10(nox)~comp*equi,data=nox)
## get prediction for the first term, comp
comp.pred <- quickPredict(nox.fit)
## plot result with method plot for quickPredict objects
plot(comp.pred)
## get prediction for the second term, equi using the binary version
equi.pred <- nox.fit
plot(equi.pred)
## get prediction for the interaction term, comp:equi
comp.equi.pred <- nox.fit
## use image method image
image(comp.equi.pred)
## use contour method
contour(comp.equi.pred,col=2,lwd=2,labcex=1.5)
contour(comp.equi.pred,what="sd",lty=3,labcex=1.2,add=TRUE)
## use persp method
persp(comp.equi.pred,theta=-10,phi=20)

## 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 <a href="#">plot</a> .
ylim	a numeric. See <a href="#">plot</a> .
do.points	see <a href="#">plot.stepfun</a> .
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 the bottom of the plot? See <a href="#">rug</a> .
...	additional arguments passed to <a href="#">plot</a> , see <a href="#">plot</a> and <a href="#">plot.stepfun</a> .

**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](mailto: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.ssanova	<i>A Plot Method for ssanova and ssanova0 Objects Tailored to Their Use in STAR</i>
--------------	---

---

**Description**

Plot a ssanova or a ssanova0 object.

**Usage**

```
## S3 method for class 'ssanova'
plot(x, y, include, ask = FALSE, ncol = 2, nrow = 3, ...)
## S3 method for class 'ssanova0'
plot(x, y, include, ask = FALSE, ncol = 2, nrow = 3, ...)
```

**Arguments**

x	a <a href="#">ssanova</a> or a <a href="#">ssanova0</a> object.
y	not used, only included for compatibility with generic method.
include	a character string with the model terms one wants to plot. If missing all terms are plotted.
ask	a logical. If TRUE terms are plotted (on a common y scale) one after the other and the user is invited to hit the enter key to generate the next plot. If FALSE (default) all terms are drawn on a suitable number of X11 devices. The number of terms on each device is controlled by arguments ncol and nrow.
ncol	the number of columns of the display matrix used on each device when ask is set to FALSE.
nrow	the number of rows of the display matrix used on each device when ask is set to FALSE.
...	not used only there for method definition compatibility.

**Value**

Nothing returned. The method is used for its side effect, plots are generated.

**Note**

The designed is inspired by the plot method for gam objects in package mgcv.

**Author(s)**

Christophe Pouzat <christophe.pouzat@gmail.com>

**See Also**

[quickPredict](#), [plot.quickPredict](#)

**Examples**

```
## Not run:
data(e060824spont)
DFA <- subset(mkGLMdf(e060824spont,0.004,0,59),neuron==1)
DFA <- within(DFA,i1 <- isi(DFA,lag=1))
DFA <- DFA[complete.cases(DFA),]
m2u1 <- mkM2U(DFA,"1N.1",0,29)
m2ui <- mkM2U(DFA,"i1",0,29,maxiter=200)
DFA <- within(DFA,e1t <- m2u1(1N.1))
DFA <- within(DFA,i1t <- m2ui(i1))
with(DFA,plot(ecdf(e1t[time>29]),pch="."))
abline(a=0,b=1,col=2,lty=2)
with(DFA,plot(ecdf(i1t[time>29]),pch="."))
abline(a=0,b=1,col=2,lty=2)
m1.fit <- gssanova(event~e1t*i1t, data=subset(DFA,time>29), family="binomial", seed=20061001)
inter.pred <- m1.fit %qp% "e1t:i1t"
contour(inter.pred,what="mean",nlevels=10,col=2,lwd=2)
contour(inter.pred,what="sd",nlevels=5,col=1,lwd=1,lty=2,add=TRUE)
inter.predN <- changeScale(inter.pred,attr(m2u1,"qFct"),attr(m2ui,"qFct"))
contour(inter.predN,what="mean",nlevels=5,col=2,lwd=1)
contour(inter.predN,what="sd",nlevels=3,col=1,lwd=1,lty=2,add=TRUE)
plot(m1.fit,nr=3,nc=1)

## 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 <code>transformedTrain</code> 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 <code>caption</code> will be used.
caption	Default caption to appear above the plots or, if <code>main</code> is given, below it
ask	logical; if TRUE, the user is <i>asked</i> before each plot, see <code>par(ask=.)</code> .
...	not used only there for compatibility with <code>plot</code> 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, Yoshihiko (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("r1N.1", "rsN.1", "rtN.1")] <- brt4df(n1.cal2sDF, "1N.1", 2, c("r1N.1", "rsN.1", "rtN.1"))
## 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(r1N.1,rsN.1,bs="cr",fx=TRUE) + rtN.1,data=n1.cal2sDF,family=binomial(link="logit"))
## 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)

```

---

predictLogProb	<i>Compute the Log Probability of a "New" Data Set Using a Fitted Model Prediction</i>
----------------	--

---

## Description

This function is designed to select models by cross-validation. If two models A and B managed to pass the Ogata's tests, one should split the data set in two parts, part1 and part2 fit each model on each part to get four fitted model objects: A1, A2, B1, B2. The chosen model should then be the one giving the largest of:  $\text{predictLogProb}(A1, \text{part2}) + \text{predictLogProb}(A2, \text{part1})$  and  $\text{predictLogProb}(B1, \text{part2}) + \text{predictLogProb}(B2, \text{part1})$ .

## Usage

```
predictLogProb(object, newdata)
```

## Arguments

object	an object inheriting from <code>ssanova</code> and <code>ssanova0</code> ( <code>gssanova</code> and <code>gssanova0</code> objects are therefore suitable).
newdata	a data frame containing the required variables. This data frame <i>must be different from the one used to obtain</i> object.

**Details**

If  $\eta[i]$  is the prediction of the fitted model for element  $i$  of `newdata` the log probability is given by :

$$\text{event}[i] * \eta[i] - \log(1 + \exp(\eta[i]))$$

Where  $\text{event}[i]$  is 0 or 1 depending on the absence or presence of a spike at the considered time. *A binomial regression is assumed here.*

**Value**

A numeric, the sum over the index  $i$  above, the log probability of the data contained in `newdata` assuming that the model contained in `object` is correct.

**Author(s)**

Christophe Pouzat <christophe.pouzat@gmail.com>

**See Also**

[mkGLMdf](#), [quickPredict](#) [gssanova](#)

**Examples**

```
## Not run:
data(e060824spont)
summary(e060824spont[["neuron 1"]])
reportHTML(e060824spont[["neuron 1"]], filename="e060824spont_1", otherST=e060824spont[c(2)], maxiter=100)
acf(diff(e060824spont[["neuron 1"]]), type="partial")
DFA <- subset(mkGLMdf(e060824spont, 0.004, 0, 59), neuron==1)
DFA <- within(DFA, i1 <- isi(DFA, lag=1))
DFA <- DFA[complete.cases(DFA), ]
m2u1 <- mkM2U(DFA, "1N.1", 0, 29, seed=20061001)
m2ui <- mkM2U(DFA, "i1", 0, 29, maxiter=200, seed=20061001)
DFA <- within(DFA, e1t <- m2u1(1N.1))
DFA <- within(DFA, i1t <- m2ui(i1))
with(DFA, plot(ecdf(e1t), pch="."))
with(DFA, plot(ecdf(i1t), pch="."))
DFAts <- as.ts(apply(DFA[, c("e1t", "i1t")], 2, qnorm))
plot(filter(DFAts, rep(1/125, 125)))
system.time(GF1 <- gssanova(event ~ e1t+i1t, data=subset(DFA, time<=29), family="binomial", seed=20061001))
tt.1 <- GF1 %tt% subset(DFA, time>29)
tt.1.summary <- summary(tt.1)
tt.1.summary
plot(tt.1.summary, which=c(1,2,4,6))
renewalTestPlot(tt.1$ppspFct())
plot(GF1, nc=1, nr=2)
system.time(GF2 <- gssanova(event ~ e1t+i1t, data=subset(DFA, time>29), family="binomial", seed=20061001))
tt.2 <- GF2 %tt% subset(DFA, time<=29)
tt.2.summary <- summary(tt.2)
tt.2.summary
plot(tt.2.summary, which=c(1,2,4,6))
renewalTestPlot(tt.2$ppspFct())
```

```

plot(GF2,nc=1,nr=2)
system.time(GF3 <- gssanova(event ~ e1t*i1t, data=subset(DFA,time<=29),family="binomial",seed=20061001))
tt.3 <- GF3 %tt% subset(DFA,time>29)
(tt.3.summary <- summary(tt.3))
plot(tt.3.summary,which=c(1,2,4,6))
renewalTestPlot(tt.3$ppspFct())
plot(GF3,nc=1,nr=3)
system.time(GF4 <- gssanova(event ~ e1t*i1t, data=subset(DFA,time>29),family="binomial",seed=20061001))
tt.4 <- GF4 %tt% subset(DFA,time<=29)
(tt.4.summary <- summary(tt.4))
plot(tt.4.summary,which=c(1,2,4,6))
renewalTestPlot(tt.4$ppspFct())
plot(GF4,nc=1,nr=3)
## Get the log probability of the data with the additive model
predictLogProb(GF1,newdata=subset(DFA,time>29))+predictLogProb(GF2,newdata=subset(DFA,time<=29))
## Get the log probability of the data with the non-additive model
predictLogProb(GF3,newdata=subset(DFA,time>29))+predictLogProb(GF4,newdata=subset(DFA,time<=29))
## The non additive model is the "best" so refit it to the whole data set
system.time(GF5 <- gssanova(event ~ e1t*i1t, data=DFA,family="binomial",seed=20061001))
plot(GF5,nr=3,nc=1)

## 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 beginning 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:

nbRepeats	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 <a href="#">round</a> ).
...	see <a href="#">print</a> and <a href="#">summary</a> .

## 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 <a href="#">hist</a> .
right	corresponding argument of <a href="#">hist</a> .
plot	corresponding argument of <a href="#">hist</a> .
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 <a href="#">plot</a> .
ylim	a numeric (default value supplied). See <a href="#">plot</a> .
xlab	a character (default value supplied). See <a href="#">plot</a> .
ylab	a character (default value supplied). See <a href="#">plot</a> .
main	a character (default value supplied). See <a href="#">plot</a> .
lwd	line width used to plot the estimated density. See <a href="#">plot</a> .
col	color used to plot the estimated density. See <a href="#">plot</a> .
...	see <a href="#">plot</a> .

### Details

When confidence bands are requested they are obtained from the quantiles 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  $i$ th bin is  $x_i$  the one of the  $(i+1)$ th bin will be  $x_i + \text{step}$ .

### 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 <a href="#">hist</a> .
mids	a numeric vector with the mid points of breaks. Similar to the component of the same name returned by <a href="#">hist</a> .
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](mailto: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	<i>Spike Trains of a Purkinje Cells (PC) Recorded in Control Conditions and With Bath Applied Bicuculline</i>
------------------	---

---

### 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  $\mu$ M 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/phycerv/physiologie\\_cerebrale.htm](http://www.biomedicale.univ-paris5.fr/phycerv/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: <a href="#">invgaussMLE</a> , <a href="#">lnormMLE</a> , <a href="#">llogisMLE</a> , <a href="#">rexpMLE</a> , <a href="#">gammaMLE</a> , <a href="#">weibullMLE</a> .
CI	a numeric vector with at most two components, the confidence intervals to be drawn. If NULL, intervals are not drawn.
type, xlab, ylab, main, sub, ylim	see <a href="#">plot</a> , 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 <a href="#">plot</a> .

**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)
```

---

quickPredict	<i>A Simple Interface to predict method for ssanova and ssanova0 objects</i>
--------------	--

---

### Description

Designed to quickly compute the effect of a *single* model term. This term can correspond to a single variable effect or to the interaction of two variables.

### Usage

```
quickPredict(object, include = object$terms$labels[2], se.fit = TRUE,
             length.out, otherTermsFct = median)
object %qp% include
```

### Arguments

object	an object inheriting from <code>ssanova</code> and <code>ssanova0</code> ( <code>gssanova</code> and <code>gssanova0</code> objects are therefore suitable).
include	a character string corresponding to a <i>single</i> model term. See <a href="#">predict.ssanova</a> and <a href="#">predict.ssanova</a> .
se.fit	logical flag indicating if standard errors are required. See <a href="#">predict.ssanova</a> and <a href="#">predict.ssanova</a> .
length.out	a positive integer, the number of points at which the prediction should be performed. These points are uniformly spread on the definition domain of the variable(s) implicitly specified by argument <code>include</code> . If missing a default of 501 for terms involving a single variable and of 101 for interaction terms involving two variables is provided.
otherTermsFct	a function applied to the other variables required for model specification.

### Details

`%qp%` is the binary version of `quickPredict`.

### Value

A `quickPredict` object. This object is a [list](#) with the following components:

xx	a numeric vector with the values of the variable specified by the model term selected by argument <code>include</code> . When an interaction term was selected the values of the first variable are stored here.
yy	a numeric vector with the values of the <i>second</i> variable specified by the <i>interaction</i> term selected by argument <code>include</code> . When selected term is not an interaction term, this component is <code>NULL</code> .
include	the value of the argument with this name.
call	the matched call.

`est.mean` a numeric vector or matrix, for intercation terms, containing the estimated mean of the term.

`est.sd` a numeric vector or matrix, for intercation terms, containing the estimated SD of the term. Is NULL is argument `se.fit` was FALSE.

**Author(s)**

Christophe Pouzat <christophe.pouzat@gmail.com>

**See Also**

[predict.ssanova](#), [predict.ssanova](#), [plot.quickPredict](#), [image.quickPredict](#), [contour.quickPredict](#), [persp.quickPredict](#), [plot.ssanova](#)

**Examples**

```
## Follow up of ssanova example of gss
data(nox)
nox.fit <- ssanova(log10(nox)~comp*equi,data=nox)
## get prediction for the first term, comp
comp.pred <- quickPredict(nox.fit)
## plot result with method plot for quickPredict objects
plot(comp.pred)
## get prediction for the second term, equi using the binary version
equi.pred <- nox.fit %qp% "equi"
plot(equi.pred)
## get prediction for the interaction term, comp:equi
comp.equi.pred <- nox.fit %qp% "comp:equi"
## use image method image
image(comp.equi.pred)
## use contour method
contour(comp.equi.pred,col=2,lwd=2,labcex=1.5)
contour(comp.equi.pred,what="sd",lty=3,labcex=1.2,add=TRUE)
## use persp method
persp(comp.equi.pred,theta=-10,phi=20)
```

---

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 <a href="#">plot</a> .
pch	data symbol used for the spikes. See <a href="#">plot</a> .
xlab	a character (default value supplied). See <a href="#">plot</a> .
ylab	a character (default value supplied). See <a href="#">plot</a> .
main	a character (default value supplied). See <a href="#">plot</a> .
...	see <a href="#">plot</a> .

**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 <a href="#">density</a> .
n	see <a href="#">density</a> .
from	see <a href="#">density</a> .
to	see <a href="#">density</a> .
na.rm	see <a href="#">density</a> .
main	see <a href="#">plot.density</a> .
xlab	see <a href="#">plot.density</a> .
ylab	see <a href="#">plot.density</a> .

type	see <code>plot.density</code> .
zero.line	see <code>plot.density</code> .
...	see <a href="#">density</a> and <code>plot.density</code> .

### 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 <a href="#">density</a> .
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](mailto: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 <code>acf.spikeTrain</code> .
d	an integer $\geq 2$ , 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 <code>chisq.test</code> .

**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 splitted 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

*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

<code>object</code>	an object from which the report is going to be generated, perhaps following some standard analysis procedure.
<code>filename</code>	a character string. The generic name of all the files (html, png as well as R data files which will be generated. See also <a href="#">HTMLInitFile</a> .
<code>extension</code>	see <a href="#">HTMLInitFile</a> .
<code>directory</code>	the full or relative path to the directory where the results are going to be stored. See also <a href="#">HTMLInitFile</a> .
<code>Title</code>	See <a href="#">HTMLInitFile</a> . If missing a default value based on filename is provided.
<code>...</code>	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>

**See Also**

[reportHTML.spikeTrain](#), [reportHTML.repeatedTrain](#), [reportHTML.gam](#)

**Examples**

```
##
```

---

reportHTML.gam	<i>Generates a Report in HTML Format from a STAR gam Object</i>
----------------	---

---

**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 <a href="#">gam</a> .
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 <a href="#">HTMLInitFile</a> .
extension	see <a href="#">HTMLInitFile</a> .
directory	the full or relative path to the directory where the results are going to be stored. See also <a href="#">HTMLInitFile</a> .
Title	See <a href="#">HTMLInitFile</a> . If missing a default value based on filename is provided.
neuron	a character string describing to which the analysis refers and used for the titles of the interaction plots (see <a href="#">plot.frt</a> ).
neuronEvts	a named list with the event variable from the data frame returned by <a href="#">mkGLMdf</a> and corresponding to the other neurons recorded simultaneously. One list element per neuron.
...	Not used, only there for compatibility 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`, interaction 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](#), [frit](#), [transformedTrain](#), [plot.transformedTrain](#), [summary.transformedTrain](#)

**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,event))
## generate the report
reportHTML(n1.spontFit1,"e070528spontN1gFit",neuron="1",neuronEvts=preN1)

## End(Not run)
```

---

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 repeatedTrain 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,
           method = c("gsspsth0", "gsspsth", "gampsth"),
           stimTimeCourse = NULL, colCI = 2,
           doTimeTransformation = TRUE, k = 100, bs = "tp",
           doGamCheck = FALSE, ...)
```

**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 <a href="#">HTMLInitFile</a> .
extension	see <a href="#">HTMLInitFile</a> .
directory	the full or relative path to the directory where the results are going to be stored. See also <a href="#">HTMLInitFile</a> .
Title	See <a href="#">HTMLInitFile</a> . If missing a default value based on filename is provided.
binSize	See <a href="#">gsspsth</a> , <a href="#">gsspsth0</a> , <a href="#">gampsth</a> .
method	A character string, the name of the function used to generate the smooth psth, one of: <a href="#">gsspsth</a> , <a href="#">gsspsth0</a> , <a href="#">gampsth</a> .
stimTimeCourse	See <a href="#">plot.repeatedTrain</a> and <a href="#">plot.gsspsth</a> , <a href="#">plot.gsspsth0</a> , <a href="#">plot.gampsth</a> .
colCI	See <a href="#">plot.gsspsth</a> , <a href="#">plot.gsspsth0</a> , <a href="#">plot.gampsth</a> .
doGamCheck	Should function <a href="#">gam.check</a> be used on the inhomogenous Poisson fit performed to obtain the smooth PSTH if method was set to <a href="#">gampsth</a> ?
doTimeTransformation	Should the estimated integrated intensity be used to perform a time transformation and generate Ogata's test plots?
k, bs	See <a href="#">gampsth</a> .
...	Passed to <a href="#">gsspsth</a> , <a href="#">gsspsth0</a> , <a href="#">gampsth</a> .

**Details**

A raster plot is added first to the report ([plot.transformedTrain](#)) with a smooth PSTH ([gsspsth](#), [gsspsth0](#), [gampsth](#).) superposed. The summary of the inhomogenous Poisson fit leading 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 [gsspsth](#), [gsspsth0](#), [gampsth](#) for details. A plot of the smooth PSTH with 95% CI (approximate in the case of [gampsth](#)) is added. If `doGamCheck` is set to TRUE and if method is set to `gampsth` 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 `gssanova`, `gssanova0` or `gamObject` object containing the result of the `gssanova`, `gssanova0` or `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`, `gsspsth`, `gsspsth0`, `gampsth`, `transformedTrain`, `plot.transformedTrain`, `summary.transformedTrain`, `gssanova`, `gssanova0`, `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"),
           method = c("gsslockedTrain0", "gsslockedTrain", "gamlockedTrain"),
```

```
doGamCheck = FALSE, k = 100, bs = "tp",
nbEvtPerBin = 10, ...)
```

### 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 <a href="#">HTMLInitFile</a> .
extension	see <a href="#">HTMLInitFile</a> .
directory	the full or relative path to the directory where the results are going to be stored. See also <a href="#">HTMLInitFile</a> .
Title	See <a href="#">HTMLInitFile</a> . If missing a default value based on filename is provided.
forceTT	Should a time transformation be performed and the <a href="#">compModels</a> plots be generated even if none of the six renewal models fits the data?
timeUnit, digits	see <a href="#">summary.spikeTrain</a> .
otherST	a named list of spikeTrain objects from simultaneously recorded neurons or nothing.
laglim	see <a href="#">lockedTrain</a> .
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.
method	A character string, the name of the function used to generate the smooth cross-correlation histograms, one of: <a href="#">gsslockedTrain</a> , <a href="#">gsslockedTrain0</a> , <a href="#">gamlockedTrain</a> .
doGamCheck	if smooth estimates are requested and method is set to <a href="#">gamlockedTrain</a> , should function <a href="#">gam.check</a> be used on them?
k	see <a href="#">gamlockedTrain</a> .
bs	see <a href="#">gamlockedTrain</a> .
nbEvtPerBin	a number of event per bin used in a way similar to the argument with the same name in <a href="#">jpsth</a> when a binning is used for pre-processing.
...	Passed to <a href="#">gsslockedTrain</a> , <a href="#">gsslockedTrain0</a> , <a href="#">gamlockedTrain</a> .

### 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 `gssanova`, `gssanova0` or `gam` fit is printed (depending on the chosen value for argument `method`). Moreover if argument `doGamCheck` is set to `TRUE` (and if `method` is set to `gamlockedTrain`) 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 `gsslockedTrain`, `gsslockedTrain0` or `gamlockedTrain` 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.transformedTrain](#), [compModels](#), [transformedTrain](#), [plot.transformedTrain](#), [summary.transformedTrain](#), [gssanova](#), [gssanova0](#), [gam](#), [gam.check](#), [lockedTrain](#), [gsslockedTrain](#), [gsslockedTrain0](#), [gamlockedTrain](#)

## 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(-1,1)*0.25, forceTT=FALSE)
## Neuron 1 of e070528spont is exceptional in that it can be well
## described by a renewal process...
```

---

rexpMLE	<i>Maximum Likelihood Parameter Estimation of a Refractory Exponential Model with Possibly Censored Data</i>
---------	--

---

### 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 <i>uncensored</i> 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. COncidence 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>

### 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 bootstrap 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)

```

**Description**

Earthquakes data used by Yoshiko 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, Yoshiko (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")

```

---

```
summary.CountingProcessSamplePath
```

*Create and Explore Counting Process Sample Path Summaries*

---

## Description

These functions / methods are designed to test a CountingProcessSamplePath object against a uniform Poisson process with rate 1.

## Usage

```

## S3 method for class 'CountingProcessSamplePath'
summary(object, exact = TRUE,
        lag.max = NULL, d = max(c(2, sqrt(length(object$ppspFct()))/%5)), ...)
## S3 method for class 'CountingProcessSamplePath.summary'
print(x, digits = 5, ...)
## S3 method for class 'CountingProcessSamplePath.summary'
plot(x, y, which = c(1,2,6,8), main,
     caption = c(expression(paste("Uniform on ", Lambda, " Test")),
                  "Berman's Test",
                  "Log Survivor Function",
                  expression(paste(U[k+1], " vs ", U[k])),
                  "Variance vs Mean Test",
                  "Wiener Process Test",
                  "Autocorrelation Fct.",
                  "Renewal Test"),
     ask = FALSE, lag.max = NULL,
     d = max(c(2, sqrt(length(eval(x$call[[2]]$ppspFct()))/%5)),
     ...))

```

## Arguments

object	A CountingProcessSamplePath object.
exact	Should an exact Kolmogorov test be used? See <a href="#">ks.test</a> .
lag.max	See <a href="#">renewalTestPlot</a> .
d	See <a href="#">renewalTestPlot</a> .

x	A CountingProcessSamplePath.summary object.
digits	An integer, the number of digits to be used while printing summaries. See <a href="#">round</a> .
y	Not used but required for compatibility with the <a href="#">plot</a> method.
which	If a subset of the test 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, below it
ask	A logical; if TRUE, the user is <i>asked</i> to hit the return key before each plot generation, see <a href="#">par</a> (ask=.).
...	Passed to <a href="#">chisq.test</a> used internally by summary, not used in plot and print.

## Details

If the CountingProcessSamplePath object  $x$  is a realization of a homogeneous Poisson process then, *conditioned on the number of events observed*, the location of the events (jumps in  $N(t)$ ) is uniform on the 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). Component UniformGivenN of a CountingProcessSamplePath.summary list contains the p.value of the Kolmogorov test of this uniform null hypothesis. The first graph generated by the plot method displays the Kolmogorov test graphically (*i.e.*, the empirical cumulative distribution function and the null hypothesis (the diagonal). The two dotted lines on both sides of the diagonal correspond to 95 and 99% (asymptotic) confidence intervals. This is the graph shown on Fig. 9 (p 19) of Ogata (1988). Notice that the summary method allows you to compute the exact p.value.

If we write  $x_i$  the jump locations of the CountingProcessSamplePath 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 realizations of iid rvs from an exponential distribution with rate 1 and the:

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

are realizations of iid rvs from a uniform distribution on  $[0,1)$ . The second graph generated by the plot method tests this uniform distribution hypotheses with a Kolmogorov Test. This is the graph shown on Fig. 10 (p 19) of Ogata (1988) which was suggested by Berman. This is also the one of the graphs proposed by Brown et al (2002) (the other one is a Q-Q plot for the same quantities). The two dotted lines on both sides of the diagonal correspond to 95 and 99% (asymptotic) confidence intervals. Component BermanTest of a CountingProcessSamplePath.summary list contains the p.value of the Kolmogorov test of this uniform null hypothesis.

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 graph generated by the plot method, using a log scale for the ordinate. The point wise CI at 95 and 99% are also drawn (dotted lines). This is the graph shown on Fig. 12 (p 20) of Ogata (1988)

If the  $u_i$  of the second paragraph are realizations of iid uniform rvs on  $[0,1)$  then a plot of  $u_{i+1}$  vs  $u_i$  should fill uniformly the unit square  $[0,1) \times [0,1)$ . This is the fourth generated graph (the one shown

on Fig. 11 (p 20) of Ogata (1988)) by the plot method while the seventh graph shows the autocorrelation function of the  $u_i$ s. Component `RenewalTest` of a `CountingProcessSamplePath.summary` list contains a slightly more elaborated (and quantitative) version of this test summarizing the fourth graph (bottom right) generated by a call to `renewalTestPlot`. This list element is itself a list with elements: `chi2.95` (a logical), `chi2.99` (a logical) and `total` (an integer). The bounds resulting from repetitively testing a sequence of what are, under the null hypothesis, iid  $\chi^2$  rvs are obtained using Donsker's Theorem (see below). For each lag the number of degrees of freedom of the  $\chi^2$  distribution is subtracted from each  $\chi^2$  value. These centered values are then divided by their sd (assuming the null hypothesis is correct). The cumulative sum of the centered and scaled sequence is formed and is divided by the square root of the maximal lag used. This is "plugged-in" the Donsker's Theorem. The eighth graph of the plot method displays the resulting Wiener process. With the tight confidence regions of Kendall et al (2007), see below.

If the  $x_i$  are realization of a homogeneous Poisson process observed between 0 and T, 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 therefore chopped into non-overlapping windows of increasing length and the empirical variance of the event count is graphed 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 graph of the plot method. This is the graph shown on Fig. 13 (p 20) of Ogata (1988). Component `varianceTimeSummary` of a `CountingProcessSamplePath.summary` list contains a summary of this test, counting the number of events out of each band.

The last graph generated by the plot method and the companions components, `Wiener95` and `Wiener99`, of a `CountingProcessSamplePath.summary` list represent "new" tests (as far as I know). They are based on the fact that if the  $y_i$  above are realizations of iid rvs following an exponential distribution with rate 1, then the  $w_i = y_i - 1$  are realizations of iid rvs with mean 0 and variance 1. We can then form the partial sums:

$$S_n = w_1 + \dots + w_n$$

and define the random right continuous with a left-hand limit functions on [0,1]:

$$\frac{1}{\sqrt{n}} S_{[nt]}$$

These functions are realizations of a process which converges (weakly) to a Wiener process on [0,1]. The proof of this statement is a corollary of Donsker's Theorem and can be found on pp 146-147, Theorem 14.1, of Billingsley (1999). I thank Vilmos Prokaj for pointing this reference to me. What is then done is testing if the putative Wiener process is entirely within the tight boundaries defined by Kendall et al (2007) for a true Wiener process, see `crossTight`.

## Value

`summary.CountingProcessSamplePath` returns a `CountingProcessSamplePath.summary` object which is a list with the following components:

<code>UniformGivenN</code>	A numeric, the p.value of the Kolmogorov test of uniformity of the events times given the number of events.
<code>Wiener95</code>	A logical: is the scaled martingale within the tight 95% confidence band?
<code>Wiener99</code>	A logical: is the scaled martingale within the tight 99% confidence band?

BermanTest	A numeric, the p.value of the Kolmogorov test of uniformity of the scaled inter events intervals.
RenewalTest	A list with components: chi2.95, chi2.99 and total. chi2.95 resp. chi2.99 is a logical and is TRUE if the Wiener process obtained as described above is within the "tight" 95% resp. 99% confidence band of Kendall et al (2007). total gives the total number of lags. See <a href="#">renewalTestPlot</a> .
varianceTime	A <a href="#">varianceTime</a> object.
varianceTimeSummary	A numeric vector with components: total, out95 and out99. total gives the total number of window sizes explored. out95 gives the number of windows within which the variance is out of the 95% confidence band. out99 gives the number of windows within which the variance is out of the 99% confidence band. See <a href="#">varianceTime</a> .
n	An integer, the number of events.
call	The matched call.

### Acknowledgments

I thank Vilmos Prokaj for pointing out Donsker's Theorem and for indicating me the proof's location (Patrick Billingsley's book).

I also thank Olivier Faugeras and Jonathan Touboul for pointing out Donsker's theorem to me.

### Warning

If you want these tests to be meaningful *do not* apply them to the data you just used to fit your conditional intensity model.

### Note

These functions / methods are designed to replace the `summary.transformedTrain` and `plot.transformedTrain` ones. The former have a more general design.

Of course to be fully usable, these functions must be coupled to functions allowing users to fit conditional intensity models. The support for that in STAR is not complete yet but is coming soon. See for now the example below.

The end of the example below (not ran by default) shows that the coverage probability of the Wiener Process confidence bands are really good even for small (50) sample sizes.

### Author(s)

Christophe Pouzat <[christophe.pouzat@gmail.com](mailto:christophe.pouzat@gmail.com)>

### References

- Patrick Billingsley (1999) *Convergence of Probability Measures*. Wiley - Interscience.
- Brillinger, D. R. (1988) Maximum likelihood analysis of spike trains of interacting nerve cells. *Biol. Cybern.* **59**: 189–200.

- 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.
- D. R. Cox and P. A. W. Lewis (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, Yoshihiko (1988) Statistical Models for Earthquake Occurrences and Residual Analysis for Point Processes. *Journal of the American Statistical Association* **83**: 9-27.
- Johnson, D.H. (1996) Point process models of single-neuron discharges. *J. Computational Neuroscience* **3**: 275–299.
- W. S. Kendall, J.- M. Marin and C. P. Robert (2007) Brownian Confidence Bands on Monte Carlo Output. *Statistics and Computing* **17**: 1–10. Preprint available at: <http://www.ceremade.dauphine.fr/~%7Exian/kmr04.rev.pdf>

### See Also

[mkCPSP](#), [as.CPSP](#), [plot.CountingProcessSamplePath](#), [print.CountingProcessSamplePath](#), [varianceTime](#), [crossTight](#), [renewalTestPlot](#), [ks.test](#)

### Examples

```
## load one spike train data set of STAR
data(e060824spont)
## Create the CountingProcessSamplePath object
n1spt.cp <- as.CPSP(e060824spont[["neuron 1"]])
## print it
n1spt.cp
## plot it
plot(n1spt.cp)
## get the summary
## Notice the warning due to few identical interspike intervals
## leading to an inaccurate Berman's test.
summary(n1spt.cp)

## Simulate data corresponding to a renewal process with
## an inverse Gaussian ISI distribution in the spontaneous
## regime modulated by a multiplicative stimulus whose time
## course is a shifted and scaled chi2 density.
## Define the "stimulus" function
stimulus <- function(t,
                     df=5,
                     tonset=5,
                     timeFactor=5,
                     peakFactor=10) {
  dchisq((t-tonset)*timeFactor, df=df)*peakFactor
}
## Define the conditional intensity / hazard function
hFct <- function(t,
                 tlast,
                 df=5,
```

```

        tonset=5,
        timeFactor=5,
        peakFactor=10,
        mu=0.075,
        sigma2=3
    ) {

    hinvgauss(t-tlast,mu=mu,sigma2=sigma2)*exp(stimulus(t,df,tonset,timeFactor,peakFactor))
}
## define the function simulating the train with the thinning method
makeTrain <- function(tstop=10,
                      peakCI=200,
                      preTime=5,
                      df=5,
                      tonset=5,
                      timeFactor=5,
                      peakFactor=10,
                      mu=0.075,
                      sigma2=3) {

    result <- numeric(500) - preTime - .Machine$double.eps
    result.n <- 500
    result[1] <- 0
    idx <- 1
    currentTime <- result[1]
    while (currentTime < tstop+preTime) {
        currentTime <- currentTime+rexp(1,peakCI)
        p <- hFct(currentTime,
                  result[idx],
                  df=df,
                  tonset=tonset+preTime,
                  timeFactor=timeFactor,
                  peakFactor=peakFactor,
                  mu=mu,
                  sigma2=sigma2)/peakCI
        rthreshold <- runif(1)
        if (p>1) stop("Wrong peakCI")
        while(p < rthreshold) {
            currentTime <- currentTime+rexp(1,peakCI)
            p <- hFct(currentTime,
                      result[idx],
                      df=df,
                      tonset=tonset+preTime,
                      timeFactor=timeFactor,
                      peakFactor=peakFactor,
                      mu=mu,
                      sigma2=sigma2)/peakCI
            if (p>1) stop("Wrong peakCI")
            rthreshold <- runif(1)
        }
        idx <- idx+1
        if (idx > result.n) {

```

```

        result <- c(result,numeric(500)) - preTime - .Machine$double.eps
        result.n <- result.n + 500
    }
    result[idx] <- currentTime
}

result[preTime < result & result <= tstop+preTime] - preTime

}
## set the seed
set.seed(20061001)
## "make" the train
t1 <- makeTrain()
## create the corresponding CountingProcessSamplePath
## object
csp1 <- mkCPSP(t1)
## print it
csp1
## test it
csp1.summary <- summary(csp1)
csp1.summary
plot(csp1.summary)
## Define a function returning the conditional intensity function (cif)
ciFct <- function(t,
                  tlast,
                  df=5,
                  tonset=5,
                  timeFactor=5,
                  peakFactor=10,
                  mu=0.075,
                  sigma2=3
                  ) {

  sapply(t, function(x) {
    if (x <= tlast[1]) return(1/mu)
    y <- x-max(tlast[tlast<x])
    hinvgauss(y,mu=mu,sigma2=sigma2)*exp(stimulus(x,df,tonset,timeFactor,peakFactor))
  })
}

## Compute the cif of the train
tt <- seq(0,10,0.001)
lambda.true <- ciFct(tt,csp1$ppspFct())
## plot it together with the events times
## Notice that the representation is somewhat inaccurate, the cif
## is in fact a left continuous function
plot(tt,lambda.true,type="l",col=2)
rug(csp1$ppspFct())
## plot the integrated intensity function and the counting process
plot(tt,cumsum(lambda.true)*0.001,type="l",col=2)
lines(csp1)
## define a function doing the time transformation / rescaling

```

```

## by integrating the cif and returning another CountingProcessSamplePath
transformCPSP <- function(cpsp,
                          ciFct,
                          CIFct,
                          method=c("integrate","discrete"),
                          subdivisions=100,
                          ...
                          ) {

  if (!inherits(cpsp,"CountingProcessSamplePath"))
    stop("cpsp should be a CountingProcessSamplePath objet")
  st <- cpsp$ppspFct()
  n <- length(st)
  from <- cpsp$from
  to <- cpsp$to
  if (missing(CIFct)) {
    if (method[1] == "integrate") {
      lwr <- c(from,st)
      upr <- c(st,to)
      Lambda <- sapply(1:(n+1),
                      function(idx)
                        integrate(ciFct,
                                lower=lwr[idx],
                                upper=upr[idx],
                                subdivisions=subdivisions,
                                ...)$value
                        )
      Lambda <- cumsum(Lambda)
      st <- Lambda[1:n]
      from <- 0
      to <- Lambda[n+1]
    } ## End of conditional on method[1] == "integrate"
    if (method[1] == "discrete") {
      lwr <- c(from,st)
      upr <- c(st,to)
      xx <- unlist(lapply(1:(n+1),
                        function(idx) seq(lwr[idx],
                                         upr[idx],
                                         length.out=subdivisions)
                        )
                  )
      Lambda <- cumsum(ciFct(xx[-length(xx)])*diff(xx))
      Lambda <- Lambda - Lambda[1]
      st <- Lambda[(1:n)*subdivisions]
      from <- 0
      to <- Lambda[length(Lambda)]
    } ## End of conditional on method[1] == "discrete"
  } else {
    result <- CIFct(c(from,st,to))
    result <- result-result[1]
    from <- result[1]
    to <- result[n+2]
    st <- result[2:(n+1)]
  }
}

```

```

} ## End of conditional on missing(CIFct)
mkCPSP(st,from,to)
}
## transform cpsp1
cpsp1t <- transformCPSP(cpsp1,function(t) ciFct(t,cpsp1$ppspFct()))
## test it
cpsp1t.summary <- summary(cpsp1t)
cpsp1t.summary
plot(cpsp1t.summary)
## Not run:
## compare the finite sample performances of the
## Kolmogorov test (test the uniformity of the
## jump times given the number of events) with the
## ones of the new "Wiener process test"
empiricalCovProb <- function(myRates=c(10,(1:8)*25,(5:10)*50,(6:10)*100),
                             nbRep=1000,
                             exact=NULL
                             ) {

  b95 <- function(t) 0.299944595870772 + 2.34797018726827*sqrt(t)
  b99 <- function(t) 0.313071417065285 + 2.88963206734397*sqrt(t)
  result <- matrix(numeric(4*length(myRates)),nrow=4)
  colnames(result) <- paste(myRates)
  rownames(result) <- c("ks95","ks99","wp95","wp99")
  for (i in 1:length(myRates)) {
    rate <- myRates[i]
    partial <- sapply(1:nbRep,
                      function(repIdx) {
                        st <- cumsum(rexp(5*rate,rate))
                        while(max(st) < 1) st <- c(st,max(st)+cumsum(rexp(5*rate,rate)))
                        st <- st[st<=1]
                        ks <- ks.test(st,punif,exact=exact)$p.value
                        w <- (st*rate-seq(st))/sqrt(rate)
                        c(ks95=0.95 < ks,
                          ks99=0.99 < ks,
                          wp95=any(w < -b95(st) | b95(st) < w),
                          wp99=any(w < -b99(st) | b99(st) < w)
                        )
                      }
                    )
    result[,i] <- apply(partial,1,sum)
  }

  attr(result,"nbRep") <- nbRep
  attr(result,"myRates") <- myRates
  attr(result,"call") <- match.call()
  result/nbRep
}

plotCovProb <- function(covprob,ci=0.95) {

```

```

nbMax <- max(attr(covprob,"myRates"))
plot(c(0,nbMax),c(0.94,1),
     type="n",
     xlab="Expected number of Spikes",
     ylab="Empirical cov. prob.",xaxs="i",yaxs="i")
nbRep <- attr(covprob,"nbRep")
polygon(c(0,nbMax,nbMax,0),
        c(rep(qbinom((1-ci)/2,nbRep,0.95)/nbRep,2),rep(qbinom(1-(1-ci)/2,nbRep,0.95)/nbRep,2)),
        col="grey50",border=NA)
polygon(c(0,nbMax,nbMax,0),
        c(rep(qbinom((1-ci)/2,nbRep,0.99)/nbRep,2),rep(qbinom(1-(1-ci)/2,nbRep,0.99)/nbRep,2)),
        col="grey50",border=NA)
nbS <- attr(covprob,"myRates")
points(nbS,1-covprob[1,],pch=3)
points(nbS,1-covprob[2,],pch=3)
points(nbS,1-covprob[3,],pch=1)
points(nbS,1-covprob[4,],pch=1)

}
system.time(covprobA <- empiricalCovProb())
plotCovProb(covprobA)

## End(Not run)

```

---

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:

<code>uniformOnTTime</code>	A two named components vector of boolean.
<code>BermanTest</code>	A two named components vector of boolean.
<code>VarTime</code>	A named component vector with as many components as passed to <a href="#">varianceTime</a> via the <code>...</code> argument with p-values of a binomial distribution.

### Author(s)

Christophe Pouzat <christophe.pouzat@gmail.com>

### References

Ogata, Yoshihiko (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$neuron=="3",]
## remove unnecessary data
rm(cal2Sdf)
## Extract the elapsed time since the second to last and
## third to last for neuron 1. Normalise the result.
n1.cal2sDF[c("r1N.1","rsN.1","rtN.1")] <- brt4df(n1.cal2sDF,"1N.1",2,c("r1N.1","rsN.1","rtN.1"))
## load mgcv library
library(mgcv)
## fit a model with a tensorial product involving the last
## three spikes and using a cubic spline basis for the last two
## To gain time use a fixed df regression spline
n1S.fitA <- gam(event ~ te(r1N.1,rsN.1,bs="cr",fx=TRUE) + rtN.1,data=n1.cal2sDF,family=binomial(link="logit"))
## transform time
N1.Lambda <- transformedTrain(n1S.fitA)
## check out the resulting spike train using the fact
## that transformedTrain objects inherit from spikeTrain
## objects
N1.Lambda
## Use more formal checks
summary(N1.Lambda)
plot(N1.Lambda,which=c(1,2,4,5),ask=FALSE)
## Transform spike trains of neuron 2 and 3
N2.Lambda <- transformedTrain(n1S.fitA,n2.cal2sDF$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)

```

## Description

Functions `thinProcess` simulates a spike train and `mkSimFct` returns a simulating function from a `gssanova` fitted model. Ogata's thinning simulation method is used. Functions `maxIntensity`, `mkSelf` and `mkMappedI` are utility functions for the first two. Function `mkPostSimAnalysis` returns a function analysing a simulated spike train. Functions `mkSimFct` and `mkPostSimAnalysis` return functions which can in principle be safely used in parallel applications, that is, they have everything they need in their closure.

**Usage**

```
thinProcess(object, m2uFctList, trueData, formerSpikes,
            intensityMax, ...)
maxIntensity(object, dfWithTime, ...)
mkSelf(m2uSelf)
mkMappedI(m2uI, lag = 1)
mkSimFct(object, m2uFctList, trueData, formerSpikes,
          intensityMax, ...)
mkPostSimAnalysis(stList, trainNumber = 1, timeWindow,
                  objects, dfFct)
```

**Arguments**

object	A <a href="#">ssanova</a> or a <a href="#">ssanova0</a> object.
m2uFctList	A list of functions. There should be as many functions as there are "internal" variables in object. An internal variable is a variable whose value is changed by the occurrence of a spike, like the elapsed time since the last spike of the simulated neuron, the duration of a former inter spike interval of a given lag, etc. The names of the components (functions) of the list should be the names of the variables. Each function should correspond to the map to uniform function used before fitting the model.
m2uSelf	The map to uniform function used to transform the actual elapsed time since the last spike values before fitting the model.
m2uI	The map to uniform function used to transform the actual former ISI durations before fitting the model.
lag	The considered lag (integer > 0).
trueData	A data frame containing the "true data" of the simulated epoch. This is to ensure that "external" variables such as the elapsed time since the last spike of a functionally coupled neuron are available.
formerSpikes	A vector of previous spike times. This is to make the computation of former inter spike intervals possible in every case.
intensityMax	The value of the maximal intensity. If missing function <code>maxIntensity</code> is called to estimate it.
dfWithTime	A data frame with one variable named "time". The latter variable is used to obtain the bin width with which the original spike train was discretized.
stList	The list of <code>spikeTrain</code> objects with one of the trains partly simulated. A single (partly simulated) <code>spikeTrain</code> object can also be used.
trainNumber	An integer, the index of the modeled and simulated spike train in <code>stList</code> .
timeWindow	A numeric vector of length 1 or 2. This argument specifies the time domain over which the fits contained in argument <code>objects</code> was performed. It is implicitly assumed that the (partial) simulation was performed outside this time domain. When a vector of length 1 is used the fitting time domain is taken as <code>c(0, timeWindow)</code> .

objects	A list of <code>ssanova</code> or <code>ssanova0</code> objects. Each element of the list is a "model" with which analysis will be performed. A single <code>ssanova</code> or <code>ssanova0</code> object can also be used.
dfFct	A function whose argument is the same as the first argument of function <code>mkGLMdf</code> and which returns a data frame suitable for use as argument <code>newdata</code> in <code>predict.ssanova</code> .
...	Additional arguments passed to <code>optim</code> which is called internally with the "BFGS" method in function <code>maxIntensity</code> . In functions <code>thinProcess</code> and <code>mkSimFct</code> , additional argument passed to function <code>maxIntensity</code> if necessary.

### Details

Function `thinProcess` simulates a spike train with Ogata's thinning method (Ogata, 1981). The latter method required the maximal intensity of the process to be known. If such is not the case, that is, if argument `intensityMax` is left missing, a proposed maximal intensity is obtained with function `maxIntensity`. If during the simulation an actual intensity larger than the given `intensityMax` occurs, the simulation is interrupted and an error message is generated.

Function `maxIntensity` uses the central point of the variable space as its initial guess. The "BFGS" method of `optim` is used to find the maximal intensity.

Function `mkPostSimAnalysis` uses function `findGlobals` in order to find among the functions called by `dfFct` the ones which are defined in the global environment. These functions are copied in the environment (Gentleman and Ihaka, 2000) of the function returned by `mkPostSimAnalysis`. If the global environment defined function called by `dfFct` *do not* call themselves over functions defined in the global environment, the returned function can be safely used as argument `fun` of package `snow`'s `clusterApply` function.

### Value

`thinProcess` returns a `spikeTrain` object.

`maxIntensity` returns the "proposed" maximal intensity (in Hz).

`mkSelf` returns a function taking two arguments: `self(proposedtime, st)`.

`mkMappedI` returns a function taking two arguments: `function(proposedtime, st)`.

`mkSimFct` returns a function simulating a `spikeTrain` object. The simulation is done with function `thinProcess`. The returned function takes no argument. The maximal intensity required by the thinning method is stored in the closure of the returned function.

`mkPostSimAnalysis` returns a function taking a `spikeTrain` object as its single argument. This function returns a list of lists. Each list correspond to one of the models in argument `objects`. Each sub list has two components: `lpp` (the log predictive probability) and `ttt` (the time transformed train, a `CountingProcessSamplePath` object).

### Note

These functions are designed to implement a rather specific type of analysis which is exposed in the "big STAR tutorial" available at: <http://sites.google.com/site/spiketrainanalysiswith/>. The exemple below shows a "complete" analysis, more details and other exemples can be found in the big tutorial.

**Author(s)**

Christophe Pouzat <christophe.pouzat@gmail.com>

**References**

Gentleman, R. and Ihaka, R. (2000) Lexical Scope and Statistical Computing. *Journal of Computational and Graphical Statistics* **9**: 491-508.

Ogata, Y. (1981) On Lewis' simulation method for point processes. *IEEE Transactions on Information Theory* **IT-29**: 23-31.

**See Also**

[gssanova](#), [as.spikeTrain](#), [mkGLMdf](#), [mkCPSP](#),

**Examples**

```
## Not run:
## The run times given in the sequel were measured on a laptop
## with a dual core CPU: 2x Intel Core 2 Duo CPU P9500 @ 2.53GHz
## The RAM was 4 GB large. The PC ran Ubuntu 9.04 and R-2.9.2
## compiled with a dynamically link ATLAS version of BLAS.

## Start by loading the data set into the work space.
data(e060824spont)
## Get a summary of neuron 1 spike train.
summary(e060824spont[["neuron 1"]])
## Run an automatic analysis of the train (takes ~ 4.22 s)
reportHTML(e060824spont[["neuron 1"]],filename="e060824spont_1",otherST=e060824spont[c(2)],maxiter=100)
## The renewal tests show that the discharge is not the one of
## a renewal process. The cross-correlogram shows no sign of
## coupling between the two neurons of the data set.

## Compute the partial autocorrelation function to get an idea
## of how many previous interspike intervals (ISIs) should be
## included in the model.
acf(diff(e060824spont[["neuron 1"]]),type="partial")
## The pacf plot suggests that the last ISI should be enough.
## Build the data frame.
DFA <- mkGLMdf(e060824spont[["neuron 1"]],0.004,0,59)
DFA <- within(DFA,i1 <- isi(DFA,lag=1))
DFA <- DFA[complete.cases(DFA),]

## look at the ECDF of the elapsed time since the last spike,
## that is, variable lN.1 of the data frame and of the last
## ISI (variable i1 of the data frame).
layout(matrix(1:2,nc=2))
with(DFA,plot(ecdf(lN.1),pch="."))
with(DFA,lines(range(lN.1),c(0,1),col=2,lty=2))
with(DFA,plot(ecdf(i1),pch="."))
with(DFA,lines(range(i1),c(0,1),col=2,lty=2))
## The distributions of these variables are clearly (and not
```

```

## surprisingly) non-uniform.

## Build emprirical functions mapping the two variables to uniform
## ones
m2u1 <- mkM2U(DFA,"1N.1",0,28.5)
m2ui <- mkM2U(DFA,"i1",0,28.5,maxiter=200)
DFA <- within(DFA,e1t <- m2u1(1N.1))
DFA <- within(DFA,i1t <- m2ui(i1))
## Cehck that the transformations did their job.
with(DFA,plot(ecdf(e1t),pch="."))
with(DFA,lines(range(e1t),c(0,1),col=2,lty=2))
with(DFA,plot(ecdf(i1t),pch="."))
with(DFA,lines(range(i1t),c(0,1),col=2,lty=2))

## The heavy computations to follow will be performed
## in parallel using the snow package.
library(snow)
## Define the number of slaves
nbSlaves <- 2
## Create the cluster.
cl <- makeCluster(rep("localhost",nbSlaves),type="SOCK")
## load STAR on each slave.
clusterEvalQ(cl,library(STAR))

## Define a function making a function performing the
## fit with gssanova and suitable for a parallel implementation.
## The returned function does in addition time transform the
## spike train on the time window not used for the fit.
mkPFct <- function(df=DFA) {
  df
  PFct <- function(gtime,
                  fm1a=event~e1t*i1t,
                  seed=20061001) {
    GF <- gssanova(fm1a,
                  data=subset(df, time %in% gtime),
                  family="binomial",
                  seed=seed)
    tt <- GF %tt% subset(df, !(time %in% gtime))
    list(GF=GF,tt=tt)
  }
  PFct
}

PFct1 <- mkPFct()
## Now PFct1 returns a list with two elements: the "fit object" (GF)
## and the time transformed train (tt)

## Create a list suitable as the second argument for clusterApply
gtList <- list(early=with(DFA,time[time<=29.5]),
              late=with(DFA,time[time>29.5])
            )

## Fit and test a model with interaction between the (mapped)

```

```

## elapsed time since the last spike and the (mapped) last
## ISI. This takes ~ 95 s.
GF1.e060824spont.1 <- clusterApply(cl, gtList, PFct1)

## Look at the test battery
plot(summary(GF1.e060824spont.1[[1]][[2]]),which=c(1,2,4,6))
plot(summary(GF1.e060824spont.1[[2]][[2]]),which=c(1,2,4,6))

## Fit and test a model without interaction between the (mapped)
## elapsed time since the last spike and the (mapped) last
## ISI. This takes ~ 61 s.
GF2.e060824spont.1 <- clusterApply(cl, gtList, PFct1,fmla=event ~ e1t+i1t)

## Look at the test battery
plot(summary(GF2.e060824spont.1[[1]][[2]]),which=c(1,2,4,6))
plot(summary(GF2.e060824spont.1[[2]][[2]]),which=c(1,2,4,6))

## Compute the "predictive log probability" with Model 2
## (without interaction). This takes ~ 1.6 s
(GF2.e060824spont.1.logProb <- predictLogProb(GF2.e060824spont.1[[1]][[1]],subset(DFA,time>29.5))+predictLogPro

## Compute the "predictive log probability" with Model 1
## (with interaction). This takes ~ 3.5 s
(GF1.e060824spont.1.logProb <- predictLogProb(GF1.e060824spont.1[[1]][[1]],subset(DFA,time>29.5))+predictLogPro

## Prepare the simulations using Model 1 and 2
## Define a function initializing a mrg32k3a RNG from
## the rstream package on each slave
initMRG32k3a <- function(cl) {
  clusterEvalQ(cl,library(rstream))
  invisible(clusterCall(cl,
    function() {
      cmd <- parse(text=".s <- new(\"rstream.mrg32k3a\")")
      eval(cmd,env=globalenv())
    }
  ))
  cat(paste(paste(clusterEvalQ(cl,rstream.sample(.s)),collapse=","),"\n"))
  invisible(clusterEvalQ(cl,rstream.reset(.s)))
}

## Define a function returning a list of independent and packed
## mrg32k3a rngs.
mkLecuyerList <- function(cl, ## a snow cluster
  seed,
  ...) {
  nbWorkers <- length(cl)
  lecuyerList <- vector(mode="list",length=nbWorkers)
  for (wIdx in 1:nbWorkers) {
    if (wIdx == 1) {
      if (!missing(seed)) lecuyerList[[1]] <- new("rstream.mrg32k3a",seed=seed)
      else lecuyerList[[1]] <- new("rstream.mrg32k3a")
    } else lecuyerList[[wIdx]] <- new("rstream.mrg32k3a")
  }
}

```

```

    rstream.packed(lecuyerList[[wIdx]]) <- TRUE
  }
  lecuyerList
}

## Define a function setting the uniform rng of each slave
## to one of the independent mrg32k3a rngs created by
## mkLecuyerList.
clusterSetupRSTREAM <- function(cl, ## a snow cluster
                                lecuyerList
                              ) {

  setLecuyer <- function(packedlecuyer) {
    assign("lecuyer",packedlecuyer,env=globalenv())
    cmd <- parse(text="rstream.packed(lecuyer)<-FALSE")
    eval(cmd,env=globalenv())
  }

  clusterApply(cl,lecuyerList,setLecuyer)
  clusterEvalQ(cl,rstream.RNG(lecuyer))

}

## Load package rstream on master.
library(rstream)
## Initialize mrg32k3a rngs on each slave.
initMRG32k3a(cl)
## Create the list of independent mrg32k3a rngs
## on master.
theList <- mkLecuyerList(cl,seed=rep(20061001,6))
## Set the uniform rng of each slave to one of the
## independent mrg32k3a rngs just created.
clusterSetupRSTREAM(cl,theList)

## Define a list of map to uniform functions
fList.e060824spont.1 <- list(e1t=mkSelf(m2u1), i1t=mkMappedI(m2ui))
## Define a simulating function from Model 1 fitted on the
## half of the data set.
simF1.e060824spont.1 <- mkSimFct(object=GF1.e060824spont.1[[1]][[1]],
                                m2uFctList=fList.e060824spont.1,
                                trueData=subset(DFA,time>29.5),
                                formerSpikes=with(DFA,time[event==1][time[event==1] <= 29.5])
                                )

## Define a simulating function from Model 2 fitted on the
## half of the data set.
simF2.e060824spont.1 <- mkSimFct(object=GF2.e060824spont.1[[1]][[1]],
                                m2uFctList=fList.e060824spont.1,
                                trueData=subset(DFA,time>29.5),
                                formerSpikes=with(DFA,time[event==1][time[event==1] <= 29.5])
                                )

```

```

## Define the number of simulations to carry out.
nbRep <- 100
## Simulate spike trains in parallel using Model 1.
## This takes ~ 670 s.
sim1.e060824spont.1 <- clusterApply(cl,
                                   rep(nbRep/nbSlaves,nbSlaves),
                                   function(n,SF) lapply(1:n, function(idx) SF()),
                                   SF=simF1.e060824spont.1)

## Convert the returned list of lists into a single
## big list.
sim1.e060824spont.1 <- c(sim1.e060824spont.1[[1]],
                        sim1.e060824spont.1[[2]])

## Simulate spike trains in parallel using Model 1.
## This takes ~ 425 s.
sim2.e060824spont.1 <- clusterApply(cl,
                                   rep(nbRep/nbSlaves,nbSlaves),
                                   function(n,SF) lapply(1:n, function(idx) SF()),
                                   SF=simF2.e060824spont.1)

## Convert the returned list of lists into a single
## big list.
sim2.e060824spont.1 <- c(sim2.e060824spont.1[[1]],
                        sim2.e060824spont.1[[2]])

## Define a function generating automatically the
## proper data frame from the simulated data.
mkDF.e060824spont.1 <- function(stList) {
  DF <- mkGLMdf(stList,0.004,0,59)
  DF <- within(DF,i1 <- isi(DF,lag=1))
  DF <- DF[complete.cases(DF),]
  DF <- within(DF,e1t <- m2u1(1N.1))
  DF <- within(DF,i1t <- m2ui(i1))
  DF
}

## Define a function analysis the simulated trains with
## both Model 1 and 2.
PSAFct <- mkPostSimAnalysis(e060824spont[[1]],1,29.5,list(GF1.e060824spont.1[[1]][[1]],GF2.e060824spont.1[[1]]

## Analyze the simulations done with Model 1.
## This takes ~ 400 s
sim1.e060824spont.1.psa <- clusterApply(cl,sim1.e060824spont.1,PSAFct)

## Analyze the simulations done with Model 2.
## This takes ~ 400 s
sim2.e060824spont.1.psa <- clusterApply(cl,sim2.e060824spont.1,PSAFct)

## Get the log predictive probability assuming Model 1 for
## simulations done with Model 1.
sim1.e060824spont.1.lpp1 <- sapply(sim1.e060824spont.1.psa, function(l) l[[1]]$lpp)
## Get the log predictive probability assuming Model 2 for

```

```

## simulations done with Model 1.
sim1.e060824spont.1.lpp2 <- sapply(sim1.e060824spont.1.psa, function(l) l[[2]]$lpp)
## Get the log predictive probability assuming Model 1 for
## simulations done with Model 2.
sim2.e060824spont.1.lpp1 <- sapply(sim2.e060824spont.1.psa, function(l) l[[1]]$lpp)
## Get the log predictive probability assuming Model 2 for
## simulations done with Model 2.
sim2.e060824spont.1.lpp2 <- sapply(sim2.e060824spont.1.psa, function(l) l[[2]]$lpp)

## Get the observed log predictive probability with each model.
e060824spont.1.lpp1 <- predictLogProb(GF1.e060824spont.1[[1]][[1]],subset(DFA,time>29.5))
e060824spont.1.lpp2 <- predictLogProb(GF2.e060824spont.1[[1]][[1]],subset(DFA,time>29.5))

## Get the difference of observed log predictive probabilities.
e060824spont.1.lppDiff <- e060824spont.1.lpp1 - e060824spont.1.lpp2

## Look at the correlation between the log predictive probabilities
## obtained with Model 1 and 2 with data simulated with Model 1.
plot(sim1.e060824spont.1.lpp1,sim1.e060824spont.1.lpp2,main="log prob with M2 vs log prob with M1 when M1 is true")

## Plot the ECDF of the log predictive probabilities obtained
## with Model 1 with data simulated with Model 1.
plot(ecdf(sim1.e060824spont.1.lpp1),pch=".",main="log prob with Model 1 when Model 1 is true")
## Show the observed value of this statistic.
segments(e060824spont.1.lpp1,0,e060824spont.1.lpp1,sum(sim1.e060824spont.1.lpp1 <= e060824spont.1.lpp1)/nbRep,c
segments(-1600,sum(sim1.e060824spont.1.lpp1 <= e060824spont.1.lpp1)/nbRep,e060824spont.1.lpp1,sum(sim1.e060824s
## Plot the ECDF of the log predictive probabilities obtained
## with Model 2 with data simulated with Model 1.
plot(ecdf(sim1.e060824spont.1.lpp2),pch=".",main="log prob with Model 2 when Model 1 is true")
## Show the observed value of this statistic.
segments(e060824spont.1.lpp2,0,e060824spont.1.lpp2,sum(sim1.e060824spont.1.lpp2 <= e060824spont.1.lpp2)/nbRep,c
segments(-1800,sum(sim1.e060824spont.1.lpp2 <= e060824spont.1.lpp2)/nbRep,e060824spont.1.lpp2,sum(sim1.e060824s

## Plot the ECDF of the difference of the log predictive probabilities
## obtained with data simulated with Model 1.
plot(ecdf(sim1.e060824spont.1.lpp1-sim1.e060824spont.1.lpp2),pch=".",main="log prob with M1 - log prob with M2 wh
## Show the observed value of this statistic.
segments(e060824spont.1.lppDiff,0,e060824spont.1.lppDiff,sum(sim1.e060824spont.1.lpp1-sim1.e060824spont.1.lpp2
segments(-10,sum(sim1.e060824spont.1.lpp1-sim1.e060824spont.1.lpp2<=e060824spont.1.lppDiff)/nbRep,e060824spont

## Look at the correlation between the log predictive probabilities
## obtained with Model 1 and 2 with data simulated with Model 2.
plot(sim2.e060824spont.1.lpp1,sim2.e060824spont.1.lpp2,main="log prob with M2 vs log prob with M1 when M2 is true")
## Plot the ECDF of the log predictive probabilities obtained
## with Model 1 with data simulated with Model 2.
plot(ecdf(sim2.e060824spont.1.lpp1),pch=".",main="log prob with Model 1 when Model 2 is true")
## Show the observed value of this statistic.
segments(e060824spont.1.lpp1,0,e060824spont.1.lpp1,sum(sim2.e060824spont.1.lpp1 <= e060824spont.1.lpp1)/nbRep,c
segments(-2000,sum(sim2.e060824spont.1.lpp1 <= e060824spont.1.lpp1)/nbRep,e060824spont.1.lpp1,sum(sim2.e060824s
## Plot the ECDF of the log predictive probabilities obtained
## with Model 2 with data simulated with Model 2.
plot(ecdf(sim2.e060824spont.1.lpp2),pch=".",main="log prob with Model 2 when Model 2 is true")
## Show the observed value of this statistic.

```

```

segments(e060824spont.1.lpp2,0,e060824spont.1.lpp2,sum(sim2.e060824spont.1.lpp2 <= e060824spont.1.lpp2)/nbRep,c
segments(-2000,sum(sim2.e060824spont.1.lpp2 <= e060824spont.1.lpp2)/nbRep,e060824spont.1.lpp2,sum(sim2.e060824s
## Plot the ECDF of the difference of the log predictive probabilities
## obtained with data simulated with Model 1.
## Make sure that the scale of the x axis is right.
xlim=c(min(c(-e060824spont.1.lppDiff,sim2.e060824spont.1.lpp2-sim2.e060824spont.1.lpp1)),max(sim2.e060824spont
plot(ecdf(sim2.e060824spont.1.lpp2-sim2.e060824spont.1.lpp1),pch=".",main="log prob with M2 - log prob with M1 wh
## Show the observed value of this statistic.
points(-e060824spont.1.lppDiff,0,pch=16,col=2)

## Stop the snow cluster.
stopCluster(c1)

## End(Not run)

```

---

transformedTrain	<i>Performs Time Transformation of Spike Trains Fitted with glm or gam</i>
------------------	--

---

## 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

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

## 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, Yoshihiko (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$neuron=="3",]
## remove unnecessary data
rm(cal2Sdf)
## Extract the elapsed time since the second to last and
## third to last for neuron 1. Normalise the result.
n1.cal2sDF[c("r1N.1","rsN.1","rtN.1")] <- brt4df(n1.cal2sDF,"1N.1",2,c("r1N.1","rsN.1","rtN.1"))
## load mgcv library
library(mgcv)
## fit a model with a tensorial product involving the last
## three spikes and using a cubic spline basis for the last two
## To gain time use a fixed df regression spline
n1S.fitA <- gam(event ~ te(r1N.1,rsN.1,bs="cr",fx=TRUE) + rtN.1,data=n1.cal2sDF,family=binomial(link="logit"))
## transform time
N1.Lambda <- transformedTrain(n1S.fitA)
## check out the resulting spike train using the fact
## that transformedTrain objects inherit from spikeTrain
## objects
N1.Lambda
## Use more formal checks
summary(N1.Lambda)
plot(N1.Lambda,which=c(1,2,4,5),ask=FALSE)
## Transform spike trains of neuron 2 and 3
N2.Lambda <- transformedTrain(n1S.fitA,n2.cal2sDF$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 <a href="#">plot</a> .
ylim	a numeric. See <a href="#">plot</a> .
...	see <a href="#">plot</a> .

### 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> and Chong Gu <chong@stat.purdue.edu> for a correction on the sampling variance of the variance of a normal distribution.

**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 <i>uncensored</i> observations for each value of yi; default: numeric(length(yi))+1.
shape.min	numeric, the initial guess of the minimal possible value of the shape parameter, used by optimise.
shape.max	numeric, the initial 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:

<code>estimate</code>	the estimated parameters, a named vector.
<code>se</code>	the standard errors, a named vector.
<code>logLik</code>	the log likelihood at maximum.
<code>r</code>	a function returning the log of the relative likelihood function.
<code>mll</code>	a function returning the opposite of the log likelihood function using the log of the parameters.
<code>call</code>	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$r(sh,sc)))
## plot contours using a linear scale for the parameters
## draw four contours corresponding to the following likelihood ratios:
## 0.5, 0.1, Chi2 with 2 df and p values of 0.95 and 0.99
X11(width=12,height=6)
layout(matrix(1:2,ncol=2))
contour(Shape,Scale,t(sampWBmleWBcontour),
       levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
       labels=c("log(0.5)",
               "log(0.1)",
               "-1/2*P(Chi2=0.95)",
               "-1/2*P(Chi2=0.99)"),
       xlab="shape",ylab="scale",
       main="Log Relative Likelihood Contours"
       )
points(sampWBmleWB$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 bootstrap 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)

```

---

%tt%

*Time Transformation Using a gssanova Objet*

---

### Description

Performs time transformation using a `gssanova` fit. If the model is correct, the result of the transformation should be a Poisson process with rate 1.

### Usage

```
gssObj %tt% dataFrame
```

### Arguments

<code>gssObj</code>	a <code>gssanova</code> or a <code>gssanova0</code> object.
<code>dataFrame</code>	a <code>data.frame</code> with variables corresponding to the ones used in the <code>gssanova</code> call giving rise to <code>gssObj</code> .

## Details

The binary operator applies `predict.ssanova` with the left side as the first argument and the right side as the second argument. The right side (`dataFrame`) must therefore contain the variables included in the formula used in the call giving rise to `gssObj`. The result of the `predict` method call is then transformed with an inverse logistic function or with an exponential (depending on the family argument, "binomial" or "poisson", used in the previous `gssanova` call). The cumulative sum is computed, that is, the integrated conditional intensity, and its value at the events times is returned as a `CountingProcessSamplePath` object.

## Value

A `CountingProcessSamplePath` object.

## Author(s)

Christophe Pouzat <christophe.pouzat@gmail.com>

## References

- Gu C. (2002) *Smoothing Spline ANOVA Models*. Springer.
- Brillinger, D. R. (1988) Maximum likelihood analysis of spike trains of interacting nerve cells. *Biol. Cybern.* **59**: 189–200.
- 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.
- Ogata, Yoshihiko (1988) Statistical Models for Earthquake Occurrences and Residual Analysis for Point Processes. *Journal of the American Statistical Association* **83**: 9-27.

## See Also

[gssanova](#), [predict.ssanova](#), [mkGLMdf](#), [mkCPSP](#), [summary.CountingProcessSamplePath](#)

## Examples

```
## Not run:
## load e060517spont data set
data(e060517spont)
## make a data frame using a 2 ms bin width
e060517spontDF <- mkGLMdf(e060517spont,0.002,0,60)
## Keep data relevant to neuron 3
e060517spontDFn3 <- e060517spontDF[e060517spontDF$neuron == "3",]
## Split data in an "early" and a "late" part
e060517spontDFn3e <- e060517spontDFn3[e060517spontDFn3$time <= 30,]
e060517spontDFn3l <- e060517spontDFn3[e060517spontDFn3$time > 30,]
## fit the late part with a nonparametric renewal model
e060517spontDFn3lGF <- gssanova(event ~ lN.3, data=e060517spontDFn3l,family="binomial")
## transform the time of the early part
e060517spont.n3e.tt <- e060517spontDFn3lGF %tt% e060517spontDFn3e
## Test the goodness of fit
e060517spont.n3e.tt
```

*%tt%*

165

```
summary(e060517spont.n3e.tt)  
plot(summary(e060517spont.n3e.tt))  
  
## End(Not run)
```

# Index

## \*Topic **IO**

reportHTML, 126

## \*Topic **datagen**

thinProcess, 147

## \*Topic **datasets**

cockroachAlData, 10

purkinjeCellData, 116

ShallowShocks, 134

## \*Topic **distribution**

coef.durationFit, 15

compModels, 16

crossGeneral, 18

dinvgauss, 28

dllogis, 31

drexp, 33

gammaMLE, 39

hgamma, 53

invgaussMLE, 60

isiHistFit, 70

llogisMLE, 74

lnormMLE, 77

qqDuration, 117

rexpMLE, 133  
summary.CountingProcessSamplePath,  
136

weibullMLE, 160

## \*Topic **file**

reportHTML, 126

## \*Topic **hplot**

plot.quickPredict, 100

plot.transformedTrain, 105

## \*Topic **htest**

%tt%, 163

crossGeneral, 18

isi, 68

summary.CountingProcessSamplePath,  
136

thinProcess, 147

## \*Topic **methods**

plot.quickPredict, 100

## \*Topic **models**

brt4df, 7

changeScale, 9

frt, 35

gamlockedTrain, 36

gamObj, 42

gampsth, 43

gsslockedTrain, 45

gssObj, 48

gsspsth, 49

jpsth, 71

mkAR, 83

mkDummy, 87

mkM2U, 94

plot.frt, 98

plot.quickPredict, 100

plot.ssanova, 104

plot.transformedTrain, 105

predictLogProb, 108

quickPredict, 119

reportHTML.gam, 127

reportHTML.repeatedTrain, 128

reportHTML.spikeTrain, 130

summary.transformedTrain, 145

transformedTrain, 156

## \*Topic **package**

STAR-package, 3

## \*Topic **print**

reportHTML, 126

## \*Topic **regression**

changeScale, 9

gamlockedTrain, 36

gamObj, 42

gampsth, 43

gsslockedTrain, 45

gssObj, 48

gsspsth, 49

mkAR, 83

- mkM2U, 94
- plot.ssanova, 104
- plot.transformedTrain, 105
- predictLogProb, 108
- quickPredict, 119
- reportHTML.gam, 127
- reportHTML.repeatedTrain, 128
- reportHTML.spikeTrain, 130
- summary.transformedTrain, 145
- \*Topic **smooth**
  - gamlockedTrain, 36
  - gamObj, 42
  - gampsth, 43
  - gsslockedTrain, 45
  - gssObj, 48
  - gsspsth, 49
  - plot.quickPredict, 100
  - plot.transformedTrain, 105
  - quickPredict, 119
  - reportHTML.gam, 127
  - reportHTML.repeatedTrain, 128
  - reportHTML.spikeTrain, 130
  - summary.transformedTrain, 145
- \*Topic **survival**
  - %tt%, 163
  - as.repeatedTrain, 5
  - as.spikeTrain, 6
  - hist.lockedTrain, 57
  - isi, 68
  - lockedTrain, 80
  - mkCPSP, 85
  - plot.spikeTrain, 102
  - print.repeatedTrain, 110
  - print.spikeTrain, 112
  - psth, 113
  - raster, 120
  - renewalTestPlot, 124
  - summary.CountingProcessSamplePath, 136
  - varianceTime, 158
- \*Topic **ts**
  - %tt%, 163
  - acf.spikeTrain, 3
  - as.repeatedTrain, 5
  - as.spikeTrain, 6
  - coef.durationFit, 15
  - compModels, 16
  - df4counts, 26
  - diff.spikeTrain, 27
  - dinvgauss, 28
  - dllogis, 31
  - drexp, 33
  - gammaMLE, 39
  - hgamma, 53
  - hist.lockedTrain, 57
  - invgaussMLE, 60
  - isi, 68
  - isiHistFit, 70
  - llogisMLE, 74
  - lnormMLE, 77
  - lockedTrain, 80
  - mkAR, 83
  - mkCPSP, 85
  - mkGLMdf, 88
  - mkREdf, 96
  - plot.spikeTrain, 102
  - print.repeatedTrain, 110
  - print.spikeTrain, 112
  - psth, 113
  - qqDuration, 117
  - raster, 120
  - rateEvolution, 122
  - renewalTestPlot, 124
  - rexpMLE, 133
  - summary.CountingProcessSamplePath, 136
  - thinProcess, 147
  - varianceTime, 158
  - weibullMLE, 160
  - [.spikeTrain(as.spikeTrain), 6
  - %frt%(frt), 35
  - %qp%(quickPredict), 119
  - %tt%, 69, 163
  - acf, 4, 5, 125
  - acf.spikeTrain, 3, 124, 125, 160
  - as.CPSP, 140
  - as.CPSP(mkCPSP), 85
  - as.repeatedTrain, 5, 26, 82, 89, 97, 111, 115, 121, 130
  - as.spikeTrain, 6, 6, 28, 82, 87, 89, 97, 103, 112, 123, 132, 150
  - attr, 73
  - attributes, 73, 95
  - brt4df, 7

- CAL1S (cockroachAlData), 10
- CAL1V (cockroachAlData), 10
- CAL2C (cockroachAlData), 10
- CAL2S (cockroachAlData), 10
- ceiling, 26, 72, 81, 88, 111
- changeScale, 9
- chisq.test, 111, 124, 125, 137
- cockroachAlData, 10
- coef, 15
- coef.durationFit, 15
- compModels, 15, 16, 70, 118, 131, 132
- contour, 73, 101, 102
- contour.jpsth (jpsth), 71
- contour.quickPredict, 120
- contour.quickPredict (plot.quickPredict), 100
- crossGeneral, 18
- crossTight, 138, 140
- crossTight (crossGeneral), 18
- data.frame, 8, 26, 69, 72, 88, 89, 97, 157
- density, 122, 123
- df4counts, 25
- diff, 27, 28
- diff.spikeTrain, 27
- dinvgauss, 28, 54, 60, 61
- dllogis, 31, 54, 75
- drexp, 33, 54, 133
- dssden, 95
- e060517ionon (cockroachAlData), 10
- e060517spont (cockroachAlData), 10
- e060817citron (cockroachAlData), 10
- e060817mix (cockroachAlData), 10
- e060817spont (cockroachAlData), 10
- e060817terpi (cockroachAlData), 10
- e060824cital (cockroachAlData), 10
- e060824spont (cockroachAlData), 10
- e070528citronellal (cockroachAlData), 10
- e070528spont (cockroachAlData), 10
- ecdf, 94
- environment, 44, 51
- factor, 26, 97
- findGlobals, 149
- floor, 26, 72, 81, 88, 111
- forwardsolve, 21
- frrt, 35, 99, 128, 130
- function, 8
- gam, 37, 38, 42, 45, 72, 87, 88, 107, 127, 128, 130, 132, 156
- gam.check, 45, 128–132
- gamlockedTrain, 36, 42, 73, 131, 132
- GammaDist, 40
- gammaMLE, 15, 17, 39, 61, 75, 117, 118, 133, 161
- gamObj, 42
- gamObject, 37, 42, 44, 130
- gampsth, 42, 43, 129, 130
- glm, 8, 26, 89, 107, 156, 157
- gssanova, 46–48, 50, 51, 69, 72, 84, 89, 108, 109, 119, 130, 132, 150, 163, 164
- gssanova0, 46–48, 50, 51, 72, 108, 119, 130, 132, 163
- gsslockedTrain, 45, 48, 73, 131, 132
- gsslockedTrain0, 48, 73, 131, 132
- gsslockedTrain0 (gsslockedTrain), 45
- gssObj, 48
- gsspsth, 48, 49, 129, 130
- gsspsth0, 48, 129, 130
- gsspsth0 (gsspsth), 49
- hgamma, 53
- hinvgauss, 30
- hinvgauss (hgamma), 53
- hist, 43, 44, 49, 50, 58, 70, 113, 114
- hist.lockedTrain, 37, 38, 47, 57, 73
- hllogis, 32
- hllogis (hgamma), 53
- hlnorm (hgamma), 53
- hrexp (hgamma), 53
- HTMLInitFile, 126, 127, 129, 131
- hweibull (hgamma), 53
- image, 73, 101, 102
- image.jpsth (jpsth), 71
- image.quickPredict, 120
- image.quickPredict (plot.quickPredict), 100
- integrate, 44, 51, 95
- invgaussMLE, 15, 17, 30, 40, 60, 75, 78, 117, 118, 133, 161
- is.durationFit (coef.durationFit), 15
- is.repeatedTrain, 111, 115, 121
- is.repeatedTrain (as.repeatedTrain), 5
- is.spikeTrain, 6, 28, 103, 112
- is.spikeTrain (as.spikeTrain), 6
- is.varianceTime (varianceTime), 158

- isi, 68
- isiHistFit, 70
- jpsth, 71, 131
- jpsth2df (jpsth), 71
- jsd (jpsth), 71
- ks.test, 136, 140
- lines, 19, 21, 57, 86, 101, 102
- lines.CountingProcessSamplePath (mkCPSP), 85
- lines.FirstPassageTime (crossGeneral), 18
- lines.quickPredict (plot.quickPredict), 100
- list, 119
- llogisMLE, 15, 17, 32, 61, 74, 117, 118
- lnormMLE, 15, 17, 40, 61, 75, 77, 117, 118, 133, 161
- lockedTrain, 37, 38, 46, 47, 57, 59, 73, 80, 131, 132
- logLik, 15
- logLik.durationFit (coef.durationFit), 15
- Lognormal, 30, 32, 78
- maxIntensity (thinProcess), 147
- methods, 110, 112
- mgcv, 8, 89, 157
- mkAR, 83
- mkCPSP, 85, 140, 150, 164
- mkDummy, 87
- mkGLMdf, 8, 35, 69, 84, 88, 88, 94, 95, 99, 107, 109, 127, 128, 146, 149, 150, 157, 164
- mkM2U, 9, 84, 94
- mkMappedI (thinProcess), 147
- mkPostSimAnalysis (thinProcess), 147
- mkREdf, 96, 123
- mkSelf (thinProcess), 147
- mkSimFct (thinProcess), 147
- mkTightBMtargetFct (crossGeneral), 18
- mPK (purkinjeCellData), 116
- optim, 60, 74, 77, 149
- par, 99, 106, 137
- persp, 73, 101, 102
- persp.jpsth (jpsth), 71
- persp.quickPredict, 120
- persp.quickPredict (plot.quickPredict), 100
- pinvgauss, 21
- pinvgauss (dinvgauss), 28
- pllogis (dllogis), 31
- plot, 19, 21, 37, 43, 44, 46, 50, 57, 72, 81, 86, 99, 101–103, 106, 114, 117, 121, 137, 159
- plot.CountingProcessSamplePath, 140
- plot.CountingProcessSamplePath (mkCPSP), 85
- plot.CountingProcessSamplePath.summary, 87
- plot.CountingProcessSamplePath.summary (summary.CountingProcessSamplePath), 136
- plot.density, 123
- plot.FirstPassageTime (crossGeneral), 18
- plot.frt, 35, 98, 127
- plot.gamlockedTrain, 73
- plot.gamlockedTrain (gamlockedTrain), 36
- plot.gampsth, 129
- plot.gampsth (gampsth), 43
- plot.gsslockedTrain, 73
- plot.gsslockedTrain (gsslockedTrain), 45
- plot.gsslockedTrain0, 73
- plot.gsslockedTrain0 (gsslockedTrain), 45
- plot.gsspsth, 129
- plot.gsspsth (gsspsth), 49
- plot.gsspsth0, 129
- plot.gsspsth0 (gsspsth), 49
- plot.hist.lockedTrain (hist.lockedTrain), 57
- plot.lockedTrain, 38, 47, 73
- plot.lockedTrain (lockedTrain), 80
- plot.psth, 45, 51
- plot.psth (psth), 113
- plot.quickPredict, 10, 100, 105, 120
- plot.rateEvolution (rateEvolution), 122
- plot.repeatedTrain, 6, 111, 129, 130
- plot.repeatedTrain (raster), 120
- plot.spikeTrain, 7, 87, 102, 131, 132
- plot.ssanova, 102, 104, 120
- plot.ssanova0 (plot.ssanova), 104
- plot.stepfun, 103
- plot.transformedTrain, 99, 105, 127–132,

- 145, 146, 157  
 plot.varianceTime (varianceTime), 158  
 Poisson, 114  
 predict.gam, 44  
 predict.ssanova, 50, 119, 120, 149, 164  
 predictLogProb, 108  
 prexp (drex), 33  
 print, 21, 47, 51, 81, 111, 112  
 print.CountingProcessSamplePath, 140  
 print.CountingProcessSamplePath  
   (mkCPSP), 85  
 print.CountingProcessSamplePath.summary,  
   87  
 print.CountingProcessSamplePath.summary  
   (summary.CountingProcessSamplePath),  
   136  
 print.FirstPassageTime (crossGeneral),  
   18  
 print.gam, 38, 44, 45  
 print.gamlockedTrain (gamlockedTrain),  
   36  
 print.gampsth (gampsth), 43  
 print.gsslockedTrain (gsslockedTrain),  
   45  
 print.gsslockedTrain0 (gsslockedTrain),  
   45  
 print.gsspsth (gsspsth), 49  
 print.gsspsth0 (gsspsth), 49  
 print.lockedTrain (lockedTrain), 80  
 print.repeatedTrain, 6, 110, 115, 121  
 print.spikeTrain, 7, 87, 103, 112  
 print.summary.repeatedTrain  
   (print.repeatedTrain), 110  
 psth, 6, 26, 45, 51, 111, 113, 121  
 purkinjeCellData, 116  
  
 qinvgauss (dinvgauss), 28  
 qllogis (dllogis), 31  
 qqDuration, 17, 117  
 qrex (drex), 33  
 quickPredict, 9, 10, 100–102, 105, 109, 119  
  
 raster, 6, 82, 111, 115, 120  
 rateEvolution, 96, 97, 122  
 renewalTestPlot, 5, 59, 103, 112, 124, 127,  
   131, 132, 136, 138–140, 160  
 reportHTML, 126  
 reportHTML.gam, 127, 127  
  
 reportHTML.repeatedTrain, 44, 45, 51, 127,  
   128  
 reportHTML.spikeTrain, 127, 130  
 rexpMLE, 15, 17, 34, 61, 75, 117, 118, 133  
 rinvgauss (dinvgauss), 28  
 rllogis (dllogis), 31  
 round, 85, 112, 137  
 rrex (drex), 33  
 rug, 103  
  
 s, 37, 43, 87, 88  
 ShallowShocks, 134  
 simulate, 50, 51  
 simulate.gsspsth (gsspsth), 49  
 simulate.gsspsth0 (gsspsth), 49  
 sPK (purkinjeCellData), 116  
 ssanova, 104, 108, 148, 149  
 ssanova0, 104, 108, 148, 149  
 ssden, 95  
 STAR (STAR-package), 3  
 STAR-package, 3  
 stepfun, 103, 112  
 summary, 21, 112  
 summary.CountingProcessSamplePath, 87,  
   136, 164  
 summary.FirstPassageTime  
   (crossGeneral), 18  
 summary.frt, 35  
 summary.frt (plot.frt), 98  
 summary.gam, 38, 44, 45, 127  
 summary.gamlockedTrain  
   (gamlockedTrain), 36  
 summary.gampsth (gampsth), 43  
 summary.gssanova, 47, 51  
 summary.gssanova0, 51  
 summary.gsslockedTrain  
   (gsslockedTrain), 45  
 summary.gsslockedTrain0  
   (gsslockedTrain), 45  
 summary.gsspsth (gsspsth), 49  
 summary.gsspsth0 (gsspsth), 49  
 summary.repeatedTrain, 6, 115, 121, 130  
 summary.repeatedTrain  
   (print.repeatedTrain), 110  
 summary.spikeTrain, 7, 86, 87, 103, 131, 132  
 summary.spikeTrain (print.spikeTrain),  
   112  
 summary.transformedTrain, 99, 107, 128,  
   130–132, 145, 157

te, 87, 88

thinProcess, 147

transformedTrain, 35, 99, 106, 107, 127,  
128, 130, 132, 145, 146, 156

uniroot, 95

varianceTime, 5, 59, 103, 107, 112, 125,  
138–140, 145, 146, 158

Weibull, 161

weibullMLE, 15, 17, 61, 75, 117, 118, 133, 160