Package 'UnitEvents'

May 14, 2018

Type Package

Title Unitary Events Method with Delayed Coincidence Count (MTGAUE or Permutation Method)

Version 0.0.3

Date 2018-05-14 10:06:27 +0200

Author Melisande Albert <melisande.albert@math.univ-toulouse.fr>,
Chahrazade Bahbah,
Yann Bouret <yann.bouret@gmail.com>,
Imen Chaarana <imen.chr@gmail.com>,
Julien Chevallier <julien.chevallier1@univ-grenoble-alpes.fr>,
Magalie Fromont <magalie.fromont@univ-rennes2.fr>,
Franck Grammont <grammont@unice.fr>,
H. Issarane,
Thomas Laloe <laloe@unice.fr>,
I. Mouline,
Patricia Reynaud-Bouret <reynaudb@unice.fr>,
Amel Rouis <rouis.a@chu-nice.fr>,
Christine Tuleau-Malot <malot@unice.fr>,
Khalil Zahri

Maintainer Gilles Scarella <gilles.scarella@unice.fr>

Description Unitary Events Method with Delayed Coincidence Count, with Gaussian Approximation (MTGAUE) or Permutation Method. Performs statistical independence tests between two neurons based on coincidence counts. Using C++ neuro-stat code.

License GPL (>=2)

URL https://github.com/ybouret/neuro-stat,
http://math.unice.fr/~malot/liste-MTGAUE.html,
https://sourcesup.renater.fr/docman/?group_id=3267

Imports methods

VignetteBuilder knitr

Suggests knitr

R topics documented:

`BH` *This function performs Benjamini-Hochberg procedure.*

Description

This function allows to identify the time windows detected by the MTGAUE or permutation procedure.

Usage

```
BH(alpha, p, plot=FALSE)
```

Arguments

`alpha`	double; $alpha$ value or maximum value for the false positive.
`p`	double; Array of p-values. Not necessarily ordered. Default value is runif(10).
`plot`	bool; To plot the result or not. Default value is FALSE.

Value

Like the component ix of the return value of sort function, the output is the ordering index vector containing the indices of the detected p-values in *p*, selected by Benjamini-Hochberg procedure.

See Also

`p.adjust`

Examples

```
alpha <- 0.05
p <- c(0.22, 0.25, 0.41, 0.11, 0.00932, 0.17, 0.0049, 0.01641, 0.166, 0.17)
indx <- BH(alpha, p)
#should return 2 values ( [7,5])
print('By Benjamini-Hochberg, detected indices are')
print(indx)
#
dev.new()
set.seed(364); # with default RNG
p <- runif(10)
indx <- BH(alpha, p, plot=TRUE)
if (!is.null(indx)) {
print('By Benjamini-Hochberg, detected indices are')
print(indx)
}
```

---

compute_time_windows

To create time windows spanning the whole interval of observation

---

Description

To create time windows of the same duration spanning the whole interval of observation

Usage

```
compute_time_windows(a, b, duration, spacing)
```

Arguments

a	Beginning of recordings
b	End of recordings
duration	Duration of a time window. Identical for every time window.
spacing	Value to define a shift between time windows. If spacing=duration, there is no overlap.

Value

Returns a list containing time windows.

- A: Matrix of time windows with two rows. Each column is a time window. The number of columns of A is equal to the number of time windows.
- L: Time step or spacing between the starts of two consecutive time windows. Equal to input argument spacing.
- len: length or number of time windows

Examples

```
# duration of each time window is 0.01, no overlapping
TW1 <- compute_time_windows(a=0, b=1, duration=0.01, spacing=0.01)
print(TW1)

# with overlapping
TW2 <- compute_time_windows(a=0, b=1, duration=0.1, spacing=0.05)
print(TW2)

# limit case 1
TW3 <- compute_time_windows(a=0, b=2, duration=2, spacing=0.5)
print(TW3)

# # limit case 2
TW4 <- compute_time_windows(a=0, b=1, duration=2, spacing=0.1)
print(TW4)

# # limit case 3
TW5 <- compute_time_windows(a=0, b=1, duration=0.1, spacing=0.2)
print(TW5)
```

---

counting_spikes *To compute the number of spikes for each trial.*

Description

To compute the number of spikes for each trial. It modifies the input matrix by adding an additional first column equal to the number of spikes for each trial.

Usage

```
counting_spikes(M)
```

Arguments

M	matrix; input matrix containing the spike times of a neuron without spike count.

Value

The output is the modified input matrix with an additional first column containing the number of spikes for each trial.

Examples

```
M <- matrix(c(0.5, 0.51, 1.1, 1.09, 0, 1.12), nrow=2)
M <- counting_spikes(M)
print(M)
```

DNeur *To create DataNeur from input neuron files or from given matrices.*

Description

To create DataNeur from input neuron files or from given matrices. A DataNeur is a matrix of size nneurons*ntrials-by-(nmax+1) containing spike times (>0), where nneurons is the number of neurons, ntrials is the number of trials and nmax is the maximum number of spikes for every neuron and trial. For a missing value, 0 is used to fill the matrix. The first column of the matrix is the number of spikes for each trial.

Usage

```
DNeur(linput=list("Neur1_c13.txt","Neur2_c13.txt"), DName="",
      listOptions=list())
```

Arguments

`linput`	Could be • a string as a file name containing the spike times of the neurons • a matrix containing the spike times of the neurons • a list; Either a list of two character vectors corresponding to the two file names containing the spike times of the two neurons or a list of two matrices containing the spike times of the two neurons
`DName`	string; Name of the DataNeur (used in plot captions). Default value is NULL.
`listOptions`	list; List of options to define the DataNeur. Names of the list are • removeCols Either a function or an array of column indices to be removed from the raw matrix. In case of a function, it should apply to any row of the matrix. • removeRows Either a function or an array of row indices to be removed from the raw matrix. In case of a function, it should apply to any column of the matrix. • scaling Numeric; a scalar value to rescale the matrix. The result will be M*scaling if M is the initial matrix. • shift Numeric; a scalar value to shift the matrix. The result will be M+shift if M is the initial matrix.

Value

The output is a matrix containing neuron spike times (>0), of size nneurons*ntrials-by-(nmax+1), where nneurons is the number of neurons, ntrials is the number of trials and nmax is the maximum number of spikes for every neuron and trial. For a missing value, 0 is used to fill the matrix. The first column of the matrix is the number of spikes for each trial. Warning: each neuron should have the same number of trials!

See Also

`counting_spikes`, `use_options`

Examples

```
# creation of a DataNeur from input files
fNeur1_13 <- system.file("extdata", "Neur1_c13.txt", package="UnitEvents")
fNeur2_13 <- system.file("extdata", "Neur2_c13.txt", package="UnitEvents")
## row data of the input files are used
D1 <- DNeur(list(fNeur1_13, fNeur2_13))
if (is.matrix(D1)) print(D1[c(1:5,200:204),1:10])
# creation of a DataNeur from matrices
M1 <- matrix(c(0.5, 0.51, 1.1, 1.09), nrow=2)
M2 <- M1 + 0.02
M2 <- cbind(M2, c(0, 1.24))
D2 <- DNeur(list(M1, M2))
if (is.matrix(D2))  print(D2)
# creation of a DataNeur from input files using 'removeCols' and 'scaling' options
##
##  function to remove columns (v should be considered as a row)
rmcols <- function(v) { return(v[-c(1,(length(v)-6):length(v))])}
## scaling
scaling <- 1/10000
fNeur1_13 <- system.file("extdata", "Neur1_c13.txt", package="UnitEvents")
D3 <- DNeur(linput=fNeur1_13, listOptions=list('removeCols'=rmcols,
       'scaling'=scaling))
if (is.matrix(D3)) print(D3[c(1:5),c(1:10)])
spikes.plot(D3)
# creation of a DataNeur from input files using 'removeCols' and 'scaling'
# options (another use of 'removeCols')
##
##  to remove columns (number of columns is known)
rmcols <- c(1,93:99)
## scaling
scaling <- 1/10000
fNeur1_13 <- system.file("extdata", "Neur1_c13.txt", package="UnitEvents")
D4 <- DNeur(linput=fNeur1_13, listOptions=list('removeCols'=rmcols,
       'scaling'=scaling))
if (any(D3!=D4)) {
    print('D3 and D4 differ')
} else
    print('D3 and D4 are identical')
# creation of a DataNeur as a Homogeneous Poisson Process for 2 neurons
lambda <- c(50, 100) # array of firing rates (for each neuron)
ntrials <- 10
D5 <- spikes.Poisson(2, lambda, 0, 2, ntrials)
dev.new()
spikes.plot(D5)
```

`draw_windows` *To draw time windows. To highlight detected windows after test.*

Description

To draw time windows. Called after Benjamin-Hochberg procedure. Fill and border colors and density can be modified (see `polygon` function). Default values are "yellow" for fill color, "black" for border and no density.

Usage

```
draw_windows(A, xrange, yrange, col="yellow", density="",
             border="black")
```

Arguments

A	double; Time window matrix (with two rows). For example it is the matrix of detected windows after Benjamini-Hochberg procedure
xrange	Double; Numeric vector of length two to define the x-coordinate range in the plot (could be missing).
yrange	Double; Numeric vector of length two to define the y-coordinate range in the plot (could be missing).
col	string; Fill color in the call to polygon function. Default value is "yellow".
density	numeric; Density in the call to polygon function. No density by default.
border	string; Border color in the call to polygon function. Default value is "black".

Examples

```
a=0; b=1
TW = compute_time_windows(a, b, spacing=0.1, duration=0.1)
xrange = c(a,b); yrange = c(-25,0)
draw_windows(TW, xrange, yrange, density=c(0), col="black")
draw_windows(TW, xrange, yrange, col="lightgreen", border=NA,
   density=c(11))
dev.new()
TW1 <- compute_time_windows(a=0, b=1, duration=0.1, spacing=0.2)
draw_windows(TW1)
```

---

findUE *To find Unitary Events in detected time windows with a given delay*

Description

To find Unitary Events in detected time windows with a given delay. Creation of the file 'UE.csv' containing the Unitary Events

Usage

```
findUE(A, DataNeur, delay)
```

Arguments

A	Matrix of detected time windows. Contains three rows. It should contain detected time windows after Benjamini-Hochberg procedure. It corresponds to the first output of MulTestsBH function.
DataNeur	A DataNeur e.g. the output of DNeur function. Matrix of spike times, of size nneurons*ntrials-by-(nmax+1), where nneurons is the number of neurons, ntrials is the number of trials and nmax is the maximum number of spikes for every neuron and trial. For a missing value, 0 is used to fill the matrix. The first column of the matrix is the number of spikes for each trial.
delay	double; delay is the value for delay used in coincidence detection.

Value

The output is a list with 2 elements

- N1: Matrix with five rows containing the Unitary Events of the first neuron; the first row contains detected spike times of the first neuron, the second one contains the trial index, the third one is the type of detection (-1 for a negative one, 1 for a positive one), fourth and fifth rows correspond to the beginning and the end of the time window.
- N2: Matrix with five rows containing the Unitary Events of the second neuron; identical to *N1* for the second neuron.

A .csv file 'UE.csv' (with separator ';') containing the Unitary Events is created.

Examples

```
## Not run:
TW = compute_time_windows(a=0, b=2.1, duration=1e-1, spacing=0.05)
rmcols <- function(v) { return(v[-c(1,(length(v)-6):length(v))])}
scaling <- 1/10000
fNeur1_40 <- system.file("extdata", "Neur1_c40.txt", package="UnitEvents")
fNeur2_40 <- system.file("extdata", "Neur2_c40.txt", package="UnitEvents")
DataNeur = DNeur(list(fNeur1_40, fNeur2_40),
 listOptions=list('removeCols'=rmcols, 'scaling'=scaling))
delay = 0.02; level = 0.05
AD = MulTestsBH(DataNeur, TW$A, delay, level) # AD contains detected time windows
T = findUE(AD, DataNeur, delay)
# examples of Unitary Events for the first neuron - trial 9
print(T$N1[,48:57])
# examples of Unitary Events for the second neuron - trial 9
print(T$N2[,64:73])

## End(Not run)
```

`minmaxtimes` *Function to compute the minimum and maximum time values of a given experiment.*

Description

Function to compute the minimum and maximum time values of a given experiment. It is especially useful for plotting or defining the windows to test on.

Usage

```
minmaxtimes(M)
```

Arguments

`M`	matrix; A DataNeur matrix e.g. the output of `DNeur` function - Contains spike times and their count per trial

Value

numeric; Minimum and maximum time values of a DataNeur

Examples

```
# creation of spikes
lambda <- 50 #firing rate
a <- 0; b <- 2 # start and end recording times
ntrials <- 10
DN = spikes.Poisson(1, lambda, a, b, ntrials)
print(paste('Minimum and maximum time values are',
     as.character(minmaxtimes(DN))))
```

`MulTestsBH` *This function performs multiple statistical tests to detect synchronization, either using Gaussian Approximation (MTGAUE) or permutation method. It runs the Benjamini-Hochberg procedure.*

Description

This function performs multiple statistical tests to detect synchronization (or dependence), either using Gaussian Approximation (MTGAUE) or permutation method.

Usage

```
MulTestsBH(DataNeur, A, delay, level, Rtest="all",
   iperm=FALSE, B=10000, num_threads=1, statistical_value='H',
   neurostatpath = "")
```

Arguments

`DataNeur`	A DataNeur e.g. the output of `DNeur` function. Matrix of spike times, of size nneurons*ntrials-by-(nmax+1), where nneurons is the number of neurons, ntrials is the number of trials and nmax is the maximum number of spikes for every neuron and trial. For a missing value, 0 is used to fill the matrix. The first column of the matrix is the number of spikes for each trial.
`A`	A two-row matrix containing time windows. Each column is a time window. The number of columns of A is equal to the number of time windows.
`delay`	double; delay is the value used in the coincidence count.
`level`	double; level is the alpha value used in Benjamin-Hochberg procedure.
`Rtest`	string; Rtest allows to define the type of the test. Only possible values are "all", "symmetric", "upper", "lower". Default value is "all" which means that upper and lower tests are performed and discernible.
`iperm`	Logical to select Permutation method or not. Default is FALSE: MTGAUE method, which performs more rapidly, is chosen but can be dubious if no stationarity. Permutation method takes more time but is more robust.
`B`	integer; size of Monte Carlo sample for Permutation method. Default is 10000.
`num_threads`	integer; Number of threads for neuro-stat computation. Default is 1.
`statistical_value`	string; Only possible values are "T" or "H". Default is "H". Corresponds to a centered computation or not.
`neurostatpath`	string; It allows to define the path of neuro-stat code if necessary. It could be a relative or an absolute path. Default value is empty.

Value

The output is a list with three fields

- The first field is a three-row matrix containing the detected time windows after Benjamini-Hochberg procedure. The first two rows contain time values of the detected windows and the last row contains the detection type: -1 for a negative detection, meaning that the coincidence count is smaller than the expected one for the time window (under independency), 1 for a positive detection, meaning that the coincidence count is bigger than the expected one for the time window (under independency).
- The second field is the vector of the p-values
- The third field is the vector of coincidences (or matrix of coincidences for the Permutation method)

Examples

```
## Not run:
# matrix of time windows
TW = compute_time_windows(a=0, b=2.1, 1e-1, 0.05)
# creation of spikes
lambda = 50 #firing rate
a = 0; b = 2 # times
ntrials = 100
D <- spikes.Poisson(2, lambda, a, b, ntrials)
# data for tests
level = 0.05
delay = 0.02

out = MulTestsBH(D, TW$A, delay, level)
print(ncol(out[[1]]))  #number of detected time windows
print(out[[1]][,1:3])

## End(Not run)
```

Neur1_c13 *Spike times for the 1st neuron of pair 13*

Description

Matrix of spike times. Times are in tenths of milliseconds. First and last seven columns contain the signal times.

Usage

```
data(Neur1_c13)
```

Format

matrix

Source

One of the data pairs described in the references

References

Multiple Tests based on a Gaussian Approximation of the Unitary Events method with delayed coincidence count. Tuleau-Malot C., Rouis A., Grammont F. and Reynaud-Bouret P., *Neural Computation*, **26**(7), pp1408–1454, 2014.

Surrogate Data Methods Based on a Shuffling of the Trials for Synchrony Detection: the Centering Issue. Albert M., Bouret Y., Fromont M., and Reynaud-Bouret P., *Neural Computation*, **28**(11), pp2352–2392, 2016.

`Neur1_c40` *Spike times for the 1st neuron of pair 40*

Description

Matrix of spike times. Times are in tenths of milliseconds. First and last seven columns contain the signal times.

Usage

```
data(Neur1_c40)
```

Format

matrix

Source

One of the data pairs described in the references

References

Multiple Tests based on a Gaussian Approximation of the Unitary Events method with delayed coincidence count. Tuleau-Malot C., Rouis A., Grammont F. and Reynaud-Bouret P., *Neural Computation*, **26**(7), pp1408–1454, 2014.

Surrogate Data Methods Based on a Shuffling of the Trials for Synchrony Detection: the Centering Issue. Albert M., Bouret Y., Fromont M., and Reynaud-Bouret P., *Neural Computation*, **28**(11), pp2352–2392, 2016.

`Neur2_c13` *Spike times for the 2nd neuron of pair 13*

Description

Matrix of spike times. Times are in tenths of milliseconds. First and last seven columns contain the signal times.

Usage

```
data(Neur2_c13)
```

Format

matrix

Source

One of the data pairs described in the references

References

Multiple Tests based on a Gaussian Approximation of the Unitary Events method with delayed coincidence count. Tuleau-Malot C., Rouis A., Grammont F. and Reynaud-Bouret P., *Neural Computation*, **26**(7), pp1408–1454, 2014.

Surrogate Data Methods Based on a Shuffling of the Trials for Synchrony Detection: the Centering Issue. Albert M., Bouret Y., Fromont M., and Reynaud-Bouret P., *Neural Computation*, **28**(11), pp2352–2392, 2016.

Neur2_c40 *Spike times for the 2nd neuron of pair 40*

Description

Matrix of spike times. Times are in tenths of milliseconds. First and last seven columns contain the signal times.

Usage

```
data(Neur2_c40)
```

Format

matrix

Source

One of the data pairs described in the references

References

Multiple Tests based on a Gaussian Approximation of the Unitary Events method with delayed coincidence count. Tuleau-Malot C., Rouis A., Grammont F. and Reynaud-Bouret P., *Neural Computation*, **26**(7), pp1408–1454, 2014.

Surrogate Data Methods Based on a Shuffling of the Trials for Synchrony Detection: the Centering Issue. Albert M., Bouret Y., Fromont M., and Reynaud-Bouret P., *Neural Computation*, **28**(11), pp2352–2392, 2016.

`spikes.inject` *To create a DataNeur simulating an injection, for several neurons, with or without jitter, using given arguments.*

Description

To create a DataNeur using given arguments, such as firing rate, start and end recording times, number of trials, firing rate of injection, jitter.

Usage

```
spikes.inject(nneurons, lambda, a, b, ntrials,
lambdainject, jitter=0)
```

Arguments

`nneurons`	integer; Number of neurons
`lambda`	vector of doubles; Firing rate of each neuron
`a`	double; Recording start time
`b`	double; Recording end time
`ntrials`	Number of trials. Should be the same for each neuron
`lambdainject`	double; Firing rate of the injection
`jitter`	double;

Value

The output is a matrix containing the spikes times of a neuron, of size nneurons*ntrials-by-(nmax+1), where nneurons is the number of neurons, ntrials is the number of trials and nmax is the maximum number of spikes for every neuron and trial. For a missing value, 0 is used to fill the matrix. The first column of the matrix is the number of spikes for each trial.

See Also

`DNeur`

Examples

```
lambda <- 50; lambdainject <- 100
ntrials <- 10; a <-0; b<-2
D <- spikes.inject(2, lambda, a, b, ntrials, lambdainject)
spikes.plot(D, v=1:2, 2)
dev.new()
jitter <- 2 # jitter is now different from 0
D <- spikes.inject(1, lambda, a, b, ntrials, lambdainject, jitter)
spikes.plot(D, v=1, 1)
```

`spikes.plot` *To plot neuron spikes*

Description

To plot neuron spikes. All the neurons or a part of them can be represented. Spikes of the first neuron are represented at the bottom in grey color, while spikes of the second neuron are represented at the top in green color.

Usage

```
spikes.plot(DataNeur, v, n, xrange, yrange, title='',
            col=c('gray','green3'))
```

Arguments

`DataNeur`	A DataNeur e.g. the output of `DNeur` function - Contains the matrices of spike times. May be a list or a matrix
`v`	numeric; A vector of indices corresponding to the neurons to be plotted. Should have a size less than or equal to n. If missing, it is equal to 1:*n*. If *n* is also missing, it is equal to 1.
`n`	The number of neurons used in the DataNeur (as a matrix). If missing, it is equal to the maximum value of *v*. If *v* is also missing, it is equal to 1.
`xrange`	Double; Numeric vector of length two to define the x-coordinate range in the plot (could be missing).
`yrange`	Double; Numeric vector of length two to define the y-coordinate range in the plot (could be missing).
`title`	string; Title of the figure. Default value is an empty string.
`col`	Character vector; Vector of length two to define colors for plotting spikes. Default value is c('gray', 'green3').

Examples

```
# creation of spikes
lambda <- 50 #firing rate
a <- 0; b <- 2 # start and end recording times
ntrials <- 10
D = spikes.Poisson(2, lambda, a, b, ntrials)
xrange <- minmaxtimes(D)
yrange <- c(0,2*ntrials+2)  # number of trials
title <- "Neuro"
# plot of the two neurons with default colors
spikes.plot(D, v=1:2, 2, xrange, yrange, title)
dev.new()
# plot of the two neurons with given colors
spikes.plot(D, v=1:2, 2, xrange, yrange, title, col=c('gray','black'))
# plot of the first neuron with default color
spikes.plot(D)
```

`spikes.Poisson` *To create a DataNeur simulating a Homogeneous Poisson Process, for several neurons, using given arguments.*

Description

To create a DataNeur using given arguments, such as firing rate, start and end recording times, number of trials.

Usage

```
spikes.Poisson(nneurons, lambda, a, b, ntrials)
```

Arguments

`nneurons`	integer; Number of neurons
`lambda`	vector of doubles; Firing rate of each neuron
`a`	double; Recording start time
`b`	double; Recording end time
`ntrials`	Number of trials. Should be the same for each neuron

Value

The output is a matrix containing the spikes times of a neuron, of size nneurons*ntrials-by-(nmax+1), where nneurons is the number of neurons, ntrials is the number of trials and nmax is the maximum number of spikes for every neuron and trial. For a missing value, 0 is used to fill the matrix. The first column of the matrix is the number of spikes for each trial.

See Also

`DNeur`

Examples

```
D <- spikes.Poisson(2, 50, 0, 2, 100)
spikes.plot(D, v=1:2, 2, xrange=minmaxtimes(D))
dev.new()
D <- spikes.Poisson(1, 50, 0, 2, 100)
spikes.plot(D, v=1, 1, xrange=minmaxtimes(D))
```

`UE.plot` *Function for plotting unitary events (coincident spikes and detected windows)*

Description

Function for plotting coincident spikes and detected windows

Usage

```
UE.plot(A, UE, ntrials, xrange, yrange, col=c('red','blue'))
```

Arguments

`A`	Matrix of detected time windows. Contains three rows. It should contain detected time windows after Benjamini-Hochberg procedure. It corresponds to the first output of `MulTestsBH` function.
`UE`	List of Unitary Events. Contains fields N1 (first neuron) and N2 (second neuron). *UE$N1* is a matrix with five rows. It corresponds to the output of `findUE` function.
`ntrials`	integer; Number of trials
`xrange`	Double; Numeric vector of length two to define the x-coordinate range in the plot (could be missing).
`yrange`	Double; Numeric vector of length two to define the y-coordinate range in the plot (could be missing).
`col`	Character vector; Vector of length two to define colors for plotting positive and negative Unitary Events. If only one input color is given, the same color is used for both types of UEs. Default value is c('red', 'blue').

See Also

`draw_windows`

Examples

```
TW = compute_time_windows(a=0, b=2.1, duration=1e-1, spacing=0.05)
rmcols <- function(v) { return(v[-c(1,(length(v)-6):length(v))])}
scaling <- 1/10000
fNeur1_40 <- system.file("extdata", "Neur1_c40.txt", package="UnitEvents")
fNeur2_40 <- system.file("extdata", "Neur2_c40.txt", package="UnitEvents")
DataNeur = DNeur(list(fNeur1_40, fNeur2_40),
 listOptions=list('removeCols'=rmcols, 'scaling'=scaling))
delay = 0.02; level = 0.05
ntrials = nrow(DataNeur)/2
AD = MulTestsBH(DataNeur, TW$A, delay, level) # AD contains detected time windows
T = findUE(AD[[1]], DataNeur, delay)
UE.plot(AD[[1]], T, ntrials)
```

UEdemo — *Demo of UnitEvents package*

Description

Demo of UnitEvents package. Examples of MTGAUE method and permutation are run.

Usage

```
UEdemo()
```

Value

A list of four UEs corresponding to each example

Examples

```
## Not run:
UEs = UEdemo()

## End(Not run)
```

UnitEvents — *Unitary Events Method with Delayed Coincidence Count, with Gaussian Approximation (MTGAUE) or Permutation Method.*

Description

Unitary Events method with delayed coincidence count, with Gaussian Approximation (MTGAUE) or permutation method. Currently works for two neurons.

Usage

```
UnitEvents(DataNeur, TW, delay=0.02, level=0.05, Rtest="all",
             iperm=FALSE, B=10000, num_threads=1,
             statistical_value="T", DName="Neuron",
     rasterPlot=TRUE, export=FALSE, neurostatpath="")
```

Arguments

DataNeur	A DataNeur e.g. the output of DNeur function. Matrix of spike times, of size nneurons*ntrials-by-(nmax+1), where nneurons is the number of neurons, ntrials is the number of trials and nmax is the maximum number of spikes for every neuron and trial. For a missing value, 0 is used to fill the matrix. The first column of the matrix is the number of spikes for each trial.
TW	List defining time windows, TW contains the following fields • A: Matrix of time windows with two rows. Each column is a time window. The number of columns of A is equal to the number of time windows. • L: Time step or spacing between the starts of two consecutive time windows.

- len: length or number of time windows

`delay` double; delay is the value used in coincidence count.

`level` double; level is the alpha value used in Benjamin-Hochberg procedure.

`Rtest` string; Rtest allows to define the type of the test. Only possible values are "all", "symmetric", "upper", "lower". Default value is "all" which means that upper and lower tests are performed and discernible.

`iperm` Logical to select Permutation method or not. Default is FALSE: MTGAUE method, which performs more rapidly, is chosen but can be dubious if no stationarity. Permutation method takes more time but is more robust.

`B` integer; size of Monte Carlo sample for Permutation method. Default is 10000.

`num_threads` integer; Number of threads for neuro-stat computation. Default is 1.

`statistical_value` string; Only possible values are "T" or "H". Default is "H". Corresponds to a centered computation or not.

`DName` string; title of the plot. Default is "Neuron".

`rasterPlot` bool; To display spikes in a window. Default is TRUE.

`export` bool; To export the raster to png format. Default is FALSE.

`neurostatpath` string; It allows to define the path of neuro-stat code if necessary. It could be a relative or an absolute path. Default value is empty which means that default path is used (for example '~/R/x86_64-pc-linux-gnu-library/3.2' on Linux)

Value

The output is a two-field list containing

- A: list of all detected time windows as a three-row matrix (the last row contains the detection sign, it corresponds to the output of `MulTestsBH` function)
- UE: list of Unitary Events as a five-row matrix with spike time, trial number, detection type and beginning and end of the time window for each event and for each neuron (corresponds to the output of `findUE` function)

References

Multiple Tests based on a Gaussian Approximation of the Unitary Events method with delayed coincidence count. Tuleau-Malot C., Rouis A., Grammont F. and Reynaud-Bouret P., *Neural Computation*, **26**(7), pp1408–1454, 2014.

Surrogate Data Methods Based on a Shuffling of the Trials for Synchrony Detection: the Centering Issue. Albert M., Bouret Y., Fromont M., and Reynaud-Bouret P., *Neural Computation*, **28**(11), pp2352–2392, 2016.

See Also

`DNeur`, `MulTestsBH`

Examples

```
rmcols <- function(v) { return(v[-c(1,(length(v)-6):length(v))])}
scaling <- 1/10000
fNeur1_13 <- system.file("extdata", "Neur1_c13.txt", package="UnitEvents")
fNeur2_13 <- system.file("extdata", "Neur2_c13.txt", package="UnitEvents")

DataNeur = DNeur(list(fNeur1_13, fNeur2_13),
                 listOptions=list('removeCols'=rmcols, 'scaling'=scaling))
res = UnitEvents(DataNeur = DataNeur,
                 TW = compute_time_windows(a=0, b=2.05, 1e-1, 0.05))
```

---

use_options *To modify the DataNeur depending on options.*

Description

To modify the DataNeur depending on options.

Usage

```
use_options(M, listOptions=list())
```

Arguments

M	matrix; input matrix containing the spike times of a neuron without spike count.
listOptions	list; List of options to define the DataNeur. May contain a scaling factor or a function to remove columns. Default value is the empty list.

Value

The output is the modified input matrix.

Examples

```
scaling <- 10
M <- matrix(c(0.5, 0.51, 1.1, 1.09), nrow=2)
M <- use_options(M, list(scaling=scaling))
print(M)
```

Index