

# SpikeOMatic-manual

September 17, 2004

## R topics documented:

classify.from.mixture . . . . .	1
display.detection.gui . . . . .	3
display.energy.evolution . . . . .	4
display.energy.histo . . . . .	6
display.events . . . . .	8
display.model . . . . .	10
display.neuron . . . . .	12
display.noise.correlation . . . . .	14
display.raw.data.gui . . . . .	16
display.spike.sweeps . . . . .	17
display.wilson . . . . .	18
find.spikes . . . . .	20
get.mcmc.model.para . . . . .	21
get.model.para . . . . .	24
get.model . . . . .	28
make.sweeps . . . . .	31
mcmc . . . . .	33
pre.mcmc . . . . .	35
read.configuration . . . . .	38
read.neurons . . . . .	40
reduce.and.whiten . . . . .	42
select.data.files . . . . .	44
spike.train.fwrite . . . . .	45
<b>Index</b>	<b>48</b>

---

`classify.from.mixture`

*Classify Spikes*

---

### Description

Classify individual spikes to their most likely generator neuron

### Usage

```
classify.from.mixture(spike.list = NULL, model.list = NULL)
```

## Arguments

- `spike.list` A list obtained with `reduce.and.whiten`. If nothing is given an object browser will pop up.
- `model.list` A list obtained with `get.model`. If nothing is given an object browser will pop up.

## Details

The second argument, `model.list`, specifies the Gaussian mixture used for classification. Each reduced and whitened spike is compared to each element (center) of the Gaussian mixture and attributed to the one with the largest posterior probability.

## Value

`reduce.and.whiten` function does not return anything but modifies its first argument, `spike.list`. The following components are added to it:

- `cluster` A vector with the index of the neuron to which each spike has been attributed.
- `spike.mahalanobis` A vector with the Mahalanobis distance between each spike and its neuron of origin (*i.e.*, the neuron to which it has been attributed).

## Author(s)

Christophe Pouzat

## References

Pouzat, Mazor and Laurent (2002) Using noise signature to optimize spike-sorting and to assess neuronal classification quality. *J Neurosci Methods* **122**: 43-57.

## Examples

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()
```

```
# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)

# Get your model
my.model <- get.model(spike1)

# Classify the spikes
classify.from.mixture(spike1, my.model)

## End(Not run)
```

---

display.detection.gui

*function to quickly check the result of a spike detection*

---

## Description

Displays and browses through raw data traces with superposed detected spikes via a GUI

## Usage

```
display.detection.gui(spikes = list())
```

## Arguments

**spikes** A list obtained through [find.spikes.with.template](#) or [find.spikes.no.template](#)

## Details

All the required parameters are set through a GUI interface. If no argument is given an object browser will pop-up inviting the user to select a "spike list"

## Value

Nothing is returned

## Author(s)

Christophe Pouzat

**See Also**

[find.spikes.with.template](#), [find.spikes.no.template](#)

**Examples**

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui(spike1)
## End(Not run)
```

---

display.energy.evolution

*Plots the evolution of the energy with respect to the MCMC steps*

---

**Description**

This function is one of the functions used to analyse the output of the MCMC algorithm run by [mcmc](#). Plots the evolution of the energy with respect to the MCMC steps at one or several betas. This enables the user to check for convergence.

**Usage**

```
display.energy.evolution(spike.list = NULL, beta = NULL)
```

**Arguments**

<code>spike.list</code>	A list obtained with <a href="#">mcmc</a> . If nothing is given an object browser will pop up.
<code>beta</code>	A vector of the inverse temperatures of the energy files to be read. If nothing is given, a window will pop up to ask for it.

**Details**

[display.energy.evolution](#) should be used immediately after running the MCMC algorithm [mcmc](#) and before reading the output files of this algorithm to decide from where to use the MCMC sample. In particular, this plot enables the user to choose the first step from where statistics will be performed by [read.configuration](#) and [read.neurons](#).

## Value

The function `display.energy.evolution` does not return anything, it modifies its first argument by adding the following components:

```
nrj.trace  
nrj.offset
```

: a list of scalars, each of them being the value of the first MC step at which the run began at the corresponding beta. The list has one component per beta.

The function `display.energy.evolution` plots the energy evolutions at each given beta with respect to the MC steps.

## Author(s)

Matthieu Delescluse and Christophe Pouzat

## References

Pouzat, Delescluse, Viot and Diebolt (2004) Improved Spike-Sorting By Modeling Firing Statistics and Burst-Dependent Spike Amplitude Attenuation: a Markov Chain Monte Carlo Approach. *J Neurophysiol* **91**: 2910-2928.

## See Also

[read.configuration](#), [read.neurons](#), [get.mcmc.model.para](#), [display.events](#), [display.wilson](#)

## Examples

```
## Not run:  
# Select data files  
trace.names <- select.data.files()  
  
# Read data files  
traces <- read.data.files(trace.names)  
  
# If you feel like it (you should) look at them  
display.raw.data.gui(traces)  
  
# Detect spikes with template  
spike1 <- find.spikes.with.template(traces,threshold=3)  
  
# Look at the structure of spike1  
str(spike1)  
  
# Check out what the template looks like  
plot(spike1$template)  
  
# Check out the detection  
display.detection.gui()  
  
# Cut the sweeps  
make.sweeps(spike1)  
  
# See the new components of spike1  
str(spike1)
```

```

# Look at the extracted sweeps
display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)

# Write the input files of the MCMC algorithm
pre.model <- pre.mcmc(spike1)

# See the new components of spike1
str(spike1)

# Start up the MCMC algorithm
mcmc(spike1)

# See the new components of spike1
str(spike1)

# See the energy evolution
display.energy.evolution(spike1)

# See the new components of spike1
str(spike1)

## End(Not run)

```

---

display.energy.histo *Display energy histograms for the REM*

---

## Description

This function allows the user to build energy histograms from energy traces generated by [mcmc](#) when used with several inverse temperatures, that is, with the REM.

## Usage

```
display.energy.histo(spike.list = NULL, nb.bins = 50, start.measurements = NULL)
```

## Arguments

<code>spike.list</code>	A list obtained with <a href="#">display.energy.evolution</a> . If nothing is given an object browser will pop up.
<code>nb.bins</code>	The number of bins used to construct the histograms
<code>start.measurements</code>	The MC step from which the histograms will be computed. If nothing is given, the last half of each energy trace will be used.

## Details

Nothing special, uses function `hist`.

**Value**

Nothing is returned.

**Author(s)**

Matthieu Delescluse and Christophe Pouzat

**References**

Pouzat, Delescluse, Viot and Diebolt (2004) Improved Spike-Sorting By Modeling Firing Statistics and Burst-Dependent Spike Amplitude Attenuation: a Markov Chain Monte Carlo Approach. *J Neurophysiol* **91**: 2910-2928.

**See Also**

[display.energy.evolution](#)

**Examples**

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)
```

```

# Write the input files of the MCMC algorithm
pre.model <- pre.mcmc(spike1)

# See the new components of spike1
str(spike1)

# Start up the MCMC algorithm
mcmc(spike1)

# See the new components of spike1
str(spike1)

# See the energy evolution
display.energy.evolution(spike1)

# See the new components of spike1
str(spike1)

## Get the energy histograms
display.energy.histo(spike1)

## End(Not run)

```

---

display.events

*Plots the spikes colored according to their neuron of origin*


---

## Description

This function is one of the functions used to analyse the output of the MCMC algorithm run by `mcmc`. Plots the spikes colored according to their neuron of origin once the spike-sorting has been performed and the `.configuration` file has been read. It also plots one neuron's spikes according to their state. `display.events` must be called after reading the `.configuration` by `read.configuration`.

## Usage

```
display.events(spike.list = NULL, left = NULL, right = NULL, chosen.neuron = NULL)
```

## Arguments

<code>spike.list</code>	A list obtained with <code>read.configuration</code> . If nothing is given an object browser will pop up.
<code>left</code>	The left limit of the plot, in seconds. If nothing is given, a window will pop up to ask for it.
<code>right</code>	The right limit of the plot, in seconds. If nothing is given, a window will pop up to ask for it.
<code>chosen.neuron</code>	The neuron number whose spikes are plotted and colored according to their states. If nothing is given, a window will pop up to ask for it.

## Details

The energy evolution given by [display.energy.evolution](#) enables the user to check that convergence has been reached at the first MCMC step considered to compute the most probable classification by the function [read.configuration](#). The function `display.events` must thus be called after these two functions.

## Value

The function `display.events` displays the 2 plots described above and does not return any value.

## Author(s)

Matthieu Delescluse

## References

Pouzat, Delescluse, Viot and Diebolt (2004) Improved Spike-Sorting By Modeling Firing Statistics and Burst-Dependent Spike Amplitude Attenuation: a Markov Chain Monte Carlo Approach. *J Neurophysiol* **91**: 2910-2928.

## See Also

[display.energy.evolution](#), [read.configuration](#), [get.mcmc.model.para](#), [read.neurons](#), [display.wilson](#)

## Examples

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
```

```

display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)

# Write the input files of the MCMC algorithm
pre.model <- pre.mcmc(spike1)

# See the new components of spike1
str(spike1)

# Start up the MCMC algorithm
mcmc(spike1)

# See the new components of spike1
str(spike1)

# See the energy evolution
display.energy.evolution(spike1)

# See the new components of spike1
str(spike1)

# Read the output file .configuration
read.configuration(spike1)

# See the new components of spike1
str(spike1)

# Plot the events colored according to their label
display.events(spike1)

## End(Not run)

```

---

display.model

*Quick visualization of single model's neurons*


---

## Description

After model parameter estimation with [get.model.param](#) this function generates a plot allowing comparison between the neuron's data and the fitted model parameters

## Usage

```
display.model(spike.list = NULL, model.list = NULL, neuron.index = 1, how.long = 5, nbins =
```

## Arguments

**spike.list** A list obtained with [classify.from.mixture](#). If nothing is given an object browser will pop up.

<code>model.list</code>	A list obtained with <code>get.model.para</code> . If nothing is given an object browser will pop up.
<code>neuron.index</code>	The neuron to display
<code>how.long</code>	The number of time constant to show on the amplitude dynamics plot.
<code>nbins</code>	The number of bins to use for the ISI density estimate.

### Details

A window will pop up with a 2 panels graph. The upper panel shows the histogram of the log of the isi of the chosen neuron with the predicted mixture density superimposed. The lower panel shows the Amplitude vs isi plot (for the amplitude coordinate on which the mean amplitude of the neuron's spikes is the largest) with the exponential relaxation fit.

### Value

Nothing is returned

### Author(s)

Christophe Pouzat

### References

Christophe Pouzat, Matthieu Delescluse, Pascal Viot and Jean Diebolt (2004) Improved Spike-Sorting By Modeling Firing Statistics and Burst-Dependent Spike Amplitude Attenuation: A Markov Chain Monte Carlo Approach. *J Neurophys* **91**: 2910-2928.

Pouzat (2004) Technique(s) for Spike - Sorting. q-bio.QM/0405012. <http://fr.arxiv.org/abs/q-bio.QM/0405012>

### See Also

[get.model.para](#), [get.model](#), [classify.from.mixture](#)

### Examples

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces, threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)
```

```

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)

# Get your model
my.model <- get.model(spike1)

# Classify the spikes
classify.from.mixture(spike1, my.model)

# Get an initial guess for the MCMC algorithm
momo.mcmc.start <- get.model.para(spike1)

# Visualize the first neuron
display.neuron(spike.list = spike1, model.list = momo.mcmc.start, neuron.index = 1, how.long = 5, nbir

## End(Not run)

```

---

display.neuron

*Diagnostic plots for classification of a specific neuron*


---

## Description

Generate diagnostic plots for classification of a specific neuron, allowing the user to set a threshold on the Mahalanobis distance to select spikes

## Usage

```
display.neuron(spike.list = NULL, neuron.index = 1, sampling.rate = 15000, mahalanobis.thres
```

## Arguments

spike.list	A list obtained with <code>classify.from.mixture</code> . If nothing is given an object browser will pop up.
neuron.index	The index of the chosen neuron.
sampling.rate	The sampling rate (in samples per seconds) used during data acquisition.

<code>mahalanobis.threshold</code>	A threshold on the mahalanobis distance of the spikes (only spikes with a smaller distance will be kept for the analysis displayed on the plots). If the parameter is not specified, all spikes are kept.
<code>bin.width</code>	The bin width (in sec) for the <i>Inter-Spike Interval (ISI)</i> histogram.

## Details

The first plot shows the spikes of neuron, `neuron.index`, displayed by an internal call to `display.spike.sweeps`. If a `mahalanobis.threshold` has been set, only spikes close enough to the neuron's center will be shown.

The upper panel of the second plot shows a Q-Q plot of the Mahalanobis distance of the spikes of neuron, `neuron.index` (or a sub-set of them if `mahalanobis.threshold` has been set), against a noise sample Mahalanobis distance (see, `reduce.and.whiten`). The lower panel shows *ISI* density estimates of neuron, `neuron.index`. One of the estimates is the classical histogram, the other is a Gaussian kernel estimate with the width of the kernel set by the 'direct plug-in' method of Sheather and Jones (1991), using the MASS implementation of it as well as the MASS trick (p 130) to avoid end effects at the origin.

## Value

Nothing is returned.

## Author(s)

Christophe Pouzat

## References

Pouzat, Mazor and Laurent (2002) Using noise signature to optimize spike-sorting and to assess neuronal classification quality. *J Neurosci Methods* **122**: 43-57.

Sheather and Jones (1991) A reliable data-based bandwidth selection method for kernel density estimation. *J Roy Stat Soc B* **53**: 683-690.

Venables and Ripley (2002) *Modern Applied Statistics with S*, 4th edition. New York: Springer Verlag.

## Examples

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
```

```

str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)

# Get your model
my.model <- get.model(spike1)

# Check out the first neuron
display.neuron(spike1, neuron.index = 1)

## End(Not run)

```

---

```
display.noise.correlation
```

*Plot the noise auto- and cross-correlation functions estimates*

---

## Description

The estimates of the auto- and cross-correlation functions are obtained from the covariance matrix of a noise sample.

## Usage

```
display.noise.correlation(noise = NULL, nb.sites = 4)
```

## Arguments

<code>noise</code>	A list generated by <a href="#">make.sweeps</a> or a matrix generated by <a href="#">make.spike.sweeps</a> . If nothing is given, an object browser will pop up.
<code>nb.sites</code>	The number of recording sites (used only for a matrix input).

## Details

This uses a slightly "quick and dirty" procedure to get the information by computing the noise covariance matrix of a noise sample. The auto-correlation functions are displayed on the diagonal and the cross-correlation functions (in an obvious way) on the upper part.

**Value**

Nothing is returned.

**Author(s)**

Christophe Pouzat

**References**

Pouzat, Mazor and Laurent (2002) Using noise signature to optimize spike-sorting and to assess neuronal classification quality. *J Neurosci Methods* **122**: 43-57.

Pouzat (2004) Technique(s) for Spike - Sorting. q-bio.QM/0405012. <http://fr.arxiv.org/abs/q-bio.QM/0405012>

**See Also**

[make.sweeps](#), [make.spike.sweeps](#), [make.noise.sweeps](#)

**Examples**

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)
```

```
## End(Not run)
```

---

display.raw.data.gui *function to quickly visualize raw data traces*

---

## Description

The function allows via a GUI menu an hopefully easy selection of data traces to display, of plot parameters, etc. One can use it as well to visualize data already loaded in R.

## Usage

```
display.raw.data.gui(data.matrix = NULL, size.of = NULL)
```

## Arguments

<code>data.matrix</code>	If given the data contained in this matrix will be displayed
<code>size.of</code>	If the data stored to disk are in float format the value this argument should be set to 4

## Details

The data to display do not need to be in R's work space they can be selected directly from files.

## Value

Nothing is returned

## Author(s)

Christophe Pouzat

## See Also

[select.data.files](#), [read.data.files](#), [display.detection.gui](#)

## Examples

```
## Not run:
# The simplest use
display.raw.data.gui()

# More sophisticated to visualize data already loaded
# That means we have to load them first
trace.names <- select.data.files()
traces <- read.data.files(trace.names)
display.raw.data.gui(traces)
## End(Not run)
```

---

`display.spike.sweeps` *Displays spike sweeps as well as mean spike and associated SD*

---

### Description

Generates a compound plot with some or all of the extracted spikes (on each channel), the sample mean spike and the sample SD

### Usage

```
display.spike.sweeps(spikes = NULL, nb.sites = 4, nb.to.display = 200, main.top = "Individual
```

### Arguments

`spikes` A list generated by `make.sweeps` or a matrix generated by `make.spike.sweeps`. If nothing is given, an object browser will pop up.

`nb.sites` The number of recording sites (used only for a matrix input).

`nb.to.display` The number of (randomly chosen) actual spikes to display on the upper part of the plot.

`main.top` A string with the title to appear at the top of the graph.

### Details

Although only a sub-sample is (possibly) displayed, the mean and SD are computed on the whole sample.

### Value

Nothing is returned.

### Author(s)

Christophe Pouzat

### References

Pouzat, Mazor and Laurent (2002) Using noise signature to optimize spike-sorting and to assess neuronal classification quality. *J Neurosci Methods* **122**: 43-57.

### See Also

`make.sweeps`, `make.spike.sweeps`, `make.noise.sweeps`, `display.neuron`

### Examples

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
```

```

display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)
## End(Not run)

```

---

display.wilson	<i>Displays Wilson plots</i>
----------------	------------------------------

---

## Description

Displays Wilson plots of the data in various ways depending of the input arguments

## Usage

```
display.wilson(spikes = NULL, nb.var = 4, sub.sample.size = 200, event.selection = 1:dim(spikes))
```

## Arguments

<code>spikes</code>	A list obtained with <code>make.sweeps</code> or a matrix of spike sweeps obtained with <code>make.spike.sweeps</code>
<code>nb.var</code>	The number of indices to use
<code>sub.sample.size</code>	The size of the sample to display
<code>event.selection</code>	A Logical array with selected events
<code>...</code>	The title of the plot

## Details

If no `spikes` argument is given, an object browser will pop up asking for it. If a sweep matrix is given as input, its SD is first computed, then its `nb.var` largest local maxima are located and selected as the indices used for the plot. If a list is given and no whitening and classification has been performed on it, its `spike.sweeps` component is extracted and plotted in the way just described. If the sample has been whitened (*i.e.*, a `spike.sweeps.white`

component does exist), then the `nb.var` indices with the largest SD are selected and plotted. If, moreover, classification has already been performed (*i.e.*, a cluster component does exist) the individual events are plotted with a cluster specific color and symbol.

### Value

Nothing is returned.

### Author(s)

Christophe Pouzat

### See Also

[display.spike.sweeps](#), [display.noise.correlation](#), [make.sweeps](#), [reduce.and.whiten](#)

### Examples

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Get the "classical" Wilson plot
display.wilson(spike1)
## End(Not run)
```

---

<code>find.spikes</code>	<i>functions to detect spikes (action potentials) on raw data traces</i>
--------------------------	--------------------------------------------------------------------------

---

### Description

These two functions perform spike detection on continuous raw data, one with, the other without, the use of a spike template. When a template is used, the function extracts it itself.

### Usage

```
find.spikes.with.template(trace.matrix = NULL, template.length = 60, threshold = 4, minimal
find.spikes.no.template(trace.matrix = NULL, threshold = 4, minimal.distance = 5)
```

### Arguments

<code>trace.matrix</code>	The matrix containing the raw data, if not given an object browser will pop up
<code>template.length</code>	The length, in sample points, of the template to construct and use
<code>threshold</code>	The detection threshold to use on the template filtered traces (a multiple of the SD of the whole trace)
<code>minimal.distance</code>	To be used if multiple traces (channels) are used simultaneously, the minimal distance (in sample points) for two spikes detected on to different traces to be considered as actually different

### Details

The template is constructed from the trace with the largest signal to noise ratio. It is made of the largest spike by setting first a detection threshold of 5 times the whole trace SD. The if 100 spikes or more are so detected, they are averaged together, normalized with a peak value of 1 and an average over their length of 0. If less than 100 spikes are detected, the threshold is lowered by 0.5 and the procedure repeated until at least 100 spikes are found.

### Value

A list with the following components

<code>find.spikes.call</code>	The function call used to get the result
<code>raw.data</code>	The name (or symbol) of the variable containing the raw data
<code>spike.found</code>	The number of detected spikes
<code>threshold</code>	The detection threshold given as argument
<code>minimal.distance</code>	The minimal distance given as argument
<code>spike.pos</code>	An array with the positions of the detected spikes in sample indices
<code>template</code>	An array with the extracted template if requested

**Author(s)**

Christophe Pouzat

**See Also**[display.detection.gui](#), [make.sweeps](#), [make.spike.sweeps](#), [make.noise.sweeps](#)**Examples**

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()
## End(Not run)
```

---

get.mcmc.model.para *Get a model estimate following an MCMC run*

---

**Description**

This function is one of the functions used to analyse the output of the MCMC algorithm run by [mcmc](#). This function computes the mean of all parameters of the MCMC model following an MCMC run. This model includes a description of the spike waveform dynamics (using an exponential relaxation) and of the neuron's discharge statistics (a mixture of log-normal densities).

**Usage**

```
get.mcmc.model.para(spike.list = NULL, firstStep = NULL)
```

**Arguments**

<code>spike.list</code>	A list obtained with <a href="#">read.neurons</a> . If nothing is given an object browser will pop up.
<code>firstStep</code>	The first MCMC step to consider to compute the mean of the model parameters. If nothing is given, a window will pop up to ask for it.

## Details

`get.mcmc.model.para` must be used after calling `display.energy.evolution` to check for convergence and choose the value of `firstStep`. It must also be used after calling `read.neurons` that reads the `.neurons` output file of the MCMC algorithm and adds the matrix `$neurons` of all parameters values across MCMC steps to the R object `spike.list`. This matrix contains the values of each parameter (column) at each MCMC step (line). It must be added to the R object `spike.list` before using `get.mcmc.model.para`, since this function computes the mean value of each column (*model parameter*) over the steps `firstStep` to `spike.list$number.of.steps`

## Value

`get.mcmc.model.para` returns a list containing the mean value of each parameter and the priors. This list is a list of lists. There is one component per neuron and a priors component, see below. The priors are just read from the file `spike.list$generic.input.name.priors` and added to the returned list. The returned list is the same as the list returned by the function `get.model.para`. The function `get.mcmc.model.para` does not modify the R object `spike.list`

## Neuron's list components

The neuron lists have the following components:

`neuron.type`: the neuron type. For now it's always "log-normal".

`neuron.states`: the number of states of the neurons.

`peak.max`: the vector of maximal peak amplitudes.

`delta`: the value of parameter delta.

`lambda`: the value of parameter lambda.

`scale`: a vector of scale values.

`shape`: a vector of shape values.

`transition`: a matrix of inter-state transition probabilities.

## Priors list components

The `priors` component is a list with the following components:

`P.max.max`

`P.max.min`

`delta.max`

`delta.min`

`lambda.max`

`lambda.min`

`scale.max`

`scale.min`

`shape.max`

`shape.min`

**Author(s)**

Matthieu Delescluse

**References**

Pouzat, Delescluse, Viot and Diebolt (2004) Improved Spike-Sorting By Modeling Firing Statistics and Burst-Dependent Spike Amplitude Attenuation: a Markov Chain Monte Carlo Approach. *J Neurophysiol* **91**: 2910-2928.

**See Also**

[display.energy.evolution](#), [coderead.neurons](#), [display.events](#), [display.wilson](#), [read.configuration](#)

**Examples**

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)

# Write the input files of the MCMC algorithm
pre.model <- pre.mcmc(spike1)

# See the new components of spike1
```

```

str(spike1)

# Start up the MCMC algorithm
mcmc(spike1)

# See the new components of spike1
str(spike1)

# See the energy evolution
display.energy.evolution(spike1)

# See the new components of spike1
str(spike1)

# Read the output file .configuration
read.configuration(spike1)

# See the new components of spike1
str(spike1)

# Plot the events colored according to their label
display.events(spike1)

# Read the output file .neurons
read.neurons(spike1)

# See the new components of spike1
str(spike1)

# Get the mean values of the model parameters
get.mcmc.model.para(spike1)

## End(Not run)

```

---

get.model.para	<i>Get a model estimate following an EM run before use of an MCMC run</i>
----------------	---------------------------------------------------------------------------

---

## Description

This function performs maximum likelihood estimation of the parameters of the model used by the MCMC algorithm. This model includes a description of the spike waveform dynamics (using an exponential relaxation) and of the neuron's discharge statistics (a mixture of log-normal densities). The classification obtained with previous EM ([get.model](#)) and mixture based sorting ([classify.from.mixture](#)) is used.

## Usage

```
get.model.para(spike.list = NULL, nb.states = 3, sampling.frequency = 15000, nb.iterations =
```

## Arguments

**spike.list** A list obtained with [classify.from.mixture](#). If nothing is given an object browser will pop up.

<code>nb.states</code>	The number of states ( <i>i.e.</i> , modes of the ISI density) of the neurons. If a single positive integer is given every neuron's will be estimated with the same number of states. Alternatively a vector specifying the number of states of each individual neuron can be given.
<code>sampling.frequency</code>	The sampling frequency in samples per second used during the recording.
<code>nb.iterations</code>	The number of iterations to use during the EM estimation of the mixture model of the ISI density.
<code>nb.tries</code>	The number of independent tries to perform for the EM estimation of the mixture model of the ISI density.
<code>priors</code>	A list containing the model's priors.
<code>start.time</code>	The time (in sec) at which the recording started.
<code>stop.time</code>	The time (in sec) at which the recording ended.
<code>update</code>	If <code>update</code> is TRUE, argument <code>spike.list</code> will be updated, a <code>state</code> and an <code>energy</code> components will be added. Otherwise the updated <code>spike.list</code> will be a component of the returned list.

## Details

In order to use `get.model.para`, argument `spike.list` must obviously have a component `cluster`. This component is used to extract the spike trains of individual neurons. That means that the function will give reasonably good initial guesses for the MCMC algorithm if the "rough" clustering performed by the sequence `get.model`, `classify.from.mixture` is not too bad.

Then for each neurons, two estimations are performed: an estimation of the exponential relaxation model of the amplitude dynamics and an estimation of the mixture of log-normal model of the ISI density. The exponential relaxation model requires the estimation of a *maximal peak amplitude* parameter,  $P_{max}$ , for each amplitude parameter used to describe a spike, a parameter  $\delta$  and a parameter  $\lambda$ , for the expected amplitude,  $A$ , given the isi is given by:

$$A(isi) = P_{max} \cdot (1 - \delta \cdot \exp(-\lambda \cdot isi))$$

This estimation is performed by finding first the amplitude coordinate on which the spikes are the largest. The correlation coefficient of the amplitude vs isi is then computed and its significance level obtained using function: `cor.test` with argument `alternative` set to "greater" and argument `method` set to "pearson". If a significance value smaller than 5% is obtained then function `nls` is called for the exponential relaxation model. If not, parameter  $P_{max}$  is set to average amplitude value on each amplitude coordinate,  $\delta$  is set to 0 if this value is allowed by the priors or to the smallest allowed value and  $\lambda$  is set to the largest allowed value.

The mixture of log-normal model fit is performed with an EM algorithm applied to the log of the isi. This EM algorithm estimates both the mean and SD values of the Gaussian. The classification generated by the algorithm is further used to estimate the inter-state transition matrix. The actual log-normal mixture model is:

$$\pi_{isi}(ISI = isi) = \sum_{l=1}^L \nu_l \cdot \pi_l(isi)$$

with,  $\nu_1 + \dots + \nu_L = 1$  and:

$$\pi_l(isi) = \frac{1}{isi \cdot f_l \cdot \sqrt{2\pi}} \exp\left(-\frac{1}{2} \left(\frac{\log(\frac{isi}{s_l})}{f_l}\right)^2\right)$$

where  $s_l$  is a *scale* parameter (measured in s) and  $f_l$  is a dimensionless shape parameter.

### Value

In any case `get.model.para` returns a list containing the estimated model and the priors. This list is a list of lists. There is one component per neuron and a priors component, see below.

If argument `update` is set to `TRUE` the only returned list is the model list, while the list making argument `spike.list` is modified as explained above. If `update` is set to `FALSE`, the list making argument `spike.list` is left unchanged and a "double" list is returned with two components, see below.

### Neuron's list components

The neuron lists have the following components:

`neuron.type`: the neuron type. For now it's always "log-normal".

`neuron.states`: the number of states of the neurons.

`peak.max`: the vector of maximal peak amplitudes.

`delta`: the value of parameter delta.

`lambda`: the value of parameter lambda.

`scale`: a vector of scale values.

`shape`: a vector of shape values.

`transition`: a matrix of inter-state transition probabilities.

### Priors list components

The `priors` component is a list with the following components:

`P.max.max`

`P.max.min`

`delta.max`

`delta.min`

`lambda.max`

`lambda.min`

`scale.max`

`scale.min`

`shape.max`

`shape.min`

### Components added to spike.list

The list making argument `spike.list` will have two components added to it:

`state`: the state of each spike.

`energy`: the "energy" of the spike train and the model.

### Components of the returned list with argument `update` set to `FALSE`

There are two components which are themselves list whose content has already been described:

`model`: the model parameters.

`original.list`: the modified `spike.list`.

### Author(s)

Christophe Pouzat

### References

Christophe Pouzat, Matthieu Delescluse, Pascal Viot and Jean Diebolt (2004) Improved Spike-Sorting By Modeling Firing Statistics and Burst-Dependent Spike Amplitude Attenuation: A Markov Chain Monte Carlo Approach. *J Neurophys* **91**: 2910-2928.

Pouzat (2004) Technique(s) for Spike - Sorting. q-bio.QM/0405012. <http://fr.arxiv.org/abs/q-bio.QM/0405012>

### See Also

[get.model.classify.from.mixture](#)

### Examples

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
```

```

display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)

# Get your model
my.model <- get.model(spike1)

# Classify the spikes
classify.from.mixture(spike1, my.model)

# Get an initial guess for the MCMC algorithm
momo.mcmc.start <- get.model.para(spike1)

## End(Not run)

```

---

get.model

*Function to estimate model parameters*


---

## Description

Performs parameter estimation with the EM algorithm and model comparison with the BIC criterion

## Usage

```
get.model(spike.list = NULL, tolerance.factor = 3.5, nb.clusters.min = 2, nb.clusters.max =
```

## Arguments

<code>spike.list</code>	A list obtained with <code>reduce.and.whiten</code> . If nothing is given an object browser will pop up.
<code>tolerance.factor</code>	A positive real number which is used to multiply the spike sweeps SD. Only spikes which are within +/- <code>tolerance.factor</code> times the SD of the spike sweeps average are kept for model estimation.
<code>nb.clusters.min</code>	The smallest number of clusters (or number of neurons) to consider.
<code>nb.clusters.max</code>	The largest number of clusters (or number of neurons) to consider.
<code>nb.iterations</code>	The number of EM iterations to perform for each initial guess.
<code>nb.tries</code>	The number of different initial guesses to try for each model considered.

## Details

After noise whitening it is assumed that the clouds associated with each neuron in reduced space are hyper-spherical Gaussians with a unit SD. The simplest version of the EM for Gaussian mixtures is therefore implemented.

## Value

A list is returned, with the following components:

<code>find.spikes.call</code>	A copy of the corresponding component of argument <code>spike.list</code>
<code>raw.data</code>	A copy of the corresponding component of argument <code>spike.list</code>
<code>threshold</code>	A copy of the corresponding component of argument <code>spike.list</code>
<code>minimal.distance</code>	A copy of the corresponding component of argument <code>spike.list</code>
<code>sweep.length</code>	A copy of the corresponding component of argument <code>spike.list</code>
<code>peak.location</code>	A copy of the corresponding component of argument <code>spike.list</code>
<code>nb.noise.evt</code>	A copy of the corresponding component of argument <code>spike.list</code>
<code>selected.indices</code>	A copy of the corresponding component of argument <code>spike.list</code>
<code>nb.clusters</code>	The best number of clusters (neurons), given the data (in the <i>BIC</i> sense)
<code>nb.cluster.range</code>	The range of number of clusters explored
<code>nb.trials</code>	The number of different initial guesses used for each model ( <i>i.e.</i> , for each number of neurons considered)
<code>bic</code>	A vector with the best <i>BIC</i> value obtained for each model considered
<code>centers</code>	A matrix with the clusters centers in the reduced and whitened space. The matrix has as many rows as there are neurons in the model ( <code>nb.clusters</code> ) and as many columns as there are dimensions in the reduced space
<code>frequency</code>	The frequency associated with each neuron (cluster) of the model
<code>log.likelihood</code>	The log likelihood sequence for the EM run which ended up giving the best <i>BIC</i> value
<code>tolerance.factor</code>	The factor used to extract the <i>clean sample</i> from which model estimation is performed. In short, the spikes of the original sample, <code>analyse.with.template\$spike.swe</code> (in the original space), are kept only if they fall within +/- <code>tolerance.factor</code> times <code>spike.sweeps.sd</code> of the average spike sweep. This is an easy way to get rid of the the most obvious superpositions of spikes
<code>full.mean</code>	A matrix with the clusters centers in the original space. The matrix has as many rows as there are neurons in the model ( <code>nb.clusters</code> ) and as many columns as there are dimensions in the original space

Three plots are moreover generated. The first one displays the *clean sample* in the usual way. The second displays diagnostic plots for the algorithm. The upper panel is a plot of the log-likelihood evolution of the best trial. The user should be careful here, a plateau should have been reached by the end, if this is not the case, the model estimation should be run again with a larger value or argument `nb.iterations`. The lower panel shows the *BIC* value obtained with the best trial of each number of neuron considered. The

user should make sure that the plot does not exhibit a monotonous decreasing trend, which indicates that argument `nb.clusters.max` was not chosen large enough (in the *BIC* sense). If such a trend is observed (*i.e.*, no minimum was reached) re-run the function with a larger `nb.clusters.max`. The ‘flatness’ of the curve around the minimum gives a qualitative measure of the quality of the evaluation of the number of neurons in the data sample. Usually it is rather flat, meaning the number of neuron is not very well defined, indeed by re-running the procedure one commonly see that the number of neuron reported can change by 2 or 3. That’s annoying but does not have real consequences on the whole sorting procedure because the number of spikes which end up being attributed to these badly defined neurons is very small (typically 2 to 5). The third plot shows the clusters centers on a Wilson plot in the reduced and whitened space.

### Author(s)

Christophe Pouzat

### References

Pouzat, Mazor and Laurent (2002) Using noise signature to optimize spike-sorting and to assess neuronal classification quality. *J Neurosci Methods* **122**: 43-57.

Pouzat (2004) Technique(s) for Spike - Sorting. q-bio.QM/0405012. <http://fr.arxiv.org/abs/q-bio.QM/0405012>

### See Also

[reduce.and.whiten](#), [display.wilson](#)

### Examples

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)
```

```

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)

# Get your model
my.model <- get.model(spike1)

## End(Not run)

```

---

make.sweeps

*make sweeps from raw data and detected spikes*


---

## Description

Extracts spike and noise sweeps from raw data after spike detection

## Usage

```

make.sweeps(spikes = NULL, sweep.length = 45, peak.location = 20,
nb.noise.evt = 2000)
make.spike.sweeps(spike.list = NULL, sweep.length = 45, peak.location = 20)
make.noise.sweeps(spike.list = NULL, sweep.length = 45, peak.location = 20, noise.sample.si

```

## Arguments

<code>spikes</code>	A list obtained through <code>find.spikes.with.template</code> or <code>find.spikes.no.template</code>
<code>sweep.length</code>	The total length of the sweeps to be extracted
<code>peak.location</code>	The location of the peak of the spike whit in each sweep
<code>nb.noise.evt</code>	The upper limit on the number of noise sweeps to extract
<code>spike.list</code>	A list obtained through <code>find.spikes.with.template</code> or <code>find.spikes.no.template</code>
<code>noise.sample.size</code>	The desired noise sample size

## Details

Does nothing fancy, just cuts sweeps at specified positions. The noise sweeps are *not* overlapping with the spike sweeps. If there is enough place on the raw data traces the requested number of noise sweeps will be cut regularly along the whole traces (minus the spike sweeps).

**Value**

`make.sweeps` function does not return anything but modifies its first argument. The following components are added to it:

`sweep.length` The length of the created sweeps.  
`peak.location` The location of the peak with the sweep.  
`nb.noise.evt` The actual number of noise sweeps extracted.  
`spike.sweeps` A matrix with the spike sweeps. Each row is one sweep. Samples from different recording sites are put one after the other.  
`noise.sweeps` A matrix with the noise sweeps.

`make.spike.sweeps` returns a matrix of spike sweeps. Each row is one sweep. Samples from different recording sites are put one after the other.

`make.noise.sweeps` returns a matrix of noise sweeps. Each row is one sweep. Samples from different recording sites are put one after the other.

**Author(s)**

Christophe Pouzat

**See Also**

[find.spikes.with.template](#), [find.spikes.no.template](#), [display.spike.sweeps](#), [display.noise.correl](#),  
[display.wilson](#), [reduce.and.whiten](#)

**Examples**

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)
## End(Not run)
```

---

<code>mcmc</code>	<i>Starts up the Markov Chain Monte Carlo (MCMC) algorithm</i>
-------------------	----------------------------------------------------------------

---

## Description

Starts up the Markov Chain Monte Carlo (MCMC) algorithm to perform the spike-sorting

## Usage

```
mcmc(spike.list = NULL, betas = NULL, number.of.steps = NULL, generic.output.name = NULL, a
```

## Arguments

<code>spike.list</code>	A list obtained with <a href="#">pre.mcmc</a> . If nothing is given an object browser will pop up.
<code>betas</code>	A vector of inverse temperatures ( <code>double</code> ), in decreasing order. If nothing is given, a window will pop up to ask for it.
<code>number.of.steps</code>	The number of MCMC steps to be performed ( <code>integer</code> ). If nothing is given, a window will pop up to ask for it.
<code>generic.output.name</code>	The generic name that will serve as prefix for the output files that will be written by the algorithm. If nothing is given, a window will pop up to ask for it.
<code>algorithm.options</code>	Options specifying the dynamics of the MCMC algorithm.

## Details

The last argument, `algorithm.options`, requires a precise knowledge of the MCMC algorithm (see References). If `TRUE`, a window will pop up to specify the options. If `FALSE`, these options are set to their default values.

## Value

`mcmc` function does not return anything but modifies its first argument, `spike.list`. The following components are added to it:

`generic.output.name`

`betas`

`number.of.steps`

`.configuration`

`.neurons`

`.nrj`

: this file contains the value of the energy of the spike train at each step, given its configuration and the parameters' values.

Each of these suffices ends with the string "\_betaValue" where betaValue is the value of beta at which the run has been performed. If the algorithm was performed at several betas simultaneously, these 3 output files are created for each beta, with the corresponding string "\_betaValue" at the end of their names. If the output name is the same as the input name, the program writes the new output *at the end* of the existing files, without removing their content.

## Author(s)

Matthieu Delescluse

## References

Pouzat, Delescluse, Viot and Diebolt (2004) Improved Spike-Sorting By Modeling Firing Statistics and Burst-Dependent Spike Amplitude Attenuation: a Markov Chain Monte Carlo Approach. *J Neurophysiol* **91**: 2910-2928.

## See Also

[pre.mcmc](#)

## Examples

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)
```

```

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)

# Write the input files of the MCMC algorithm
pre.model <- pre.mcmc(spike1)

# See the new components of spike1
str(spike1)

# Start up the MCMC algorithm
mcmc(spike1)

# See the new components of spike1
str(spike1)

## End(Not run)

```

---

pre.mcmc

---

*Write necessary files to disk before a MCMC run*


---

## Description

Running the MCMC algorithm requires some files to be already written to disk (with data, initial guesses, etc). This function does the necessary job.

## Usage

```
pre.mcmc(spike.list = NULL, generic.input.name = NULL, initial.model = NULL, nb.states = 3,
```

## Arguments

<code>spike.list</code>	A list obtained with <code>reduce.and.whiten</code> or <code>classify.from.mixture</code> . If nothing is given an object browser will pop up.
<code>generic.input.name</code>	The generic name used to name the files written to disk. If nothing is given, a window pops up.
<code>initial.model</code>	A list obtained with <code>get.model.para</code> .
<code>nb.states</code>	The number of states per neuron, used only if argument <code>initial.model</code> is not provided.
<code>sampling.frequency</code>	The sampling frequency in samples per second used during the recording.
<code>nb.iterations</code>	The number of iterations to use during the EM estimation of the mixture model of the ISI density, used only if argument <code>initial.model</code> is not provided.
<code>nb.tries</code>	The number of independent tries to perform for the EM estimation of the mixture model of the ISI density, used only if argument <code>initial.model</code> is not provided.

<code>priors</code>	A list containing the model's priors. Default values are used if nothing is provided and if argument <code>initial.model</code> is not provided.
<code>use.cluster</code>	Indicates if a clustering performed with EM algorithm should be used to get initial guesses.
<code>nb.neurons</code>	The number of neurons in the model. Has to be provided only if argument <code>use.cluster</code> is set to <code>FALSE</code> .
<code>start.time</code>	The time (in sec) at which the recording started.
<code>stop.time</code>	The time (in sec) at which the recording ended.

## Details

If argument `use.cluster` is set to `FALSE` initial guesses are generated at random as follows. As many real events are randomly selected from the actual events as specified by argument `nb.neurons`. They provide initial guesses for the maximal peak amplitudes of the model's neurons. After the neurons are all given the same parameters, `delta` is set to 0 if this value is allowed by the priors or to the smallest allowed value. `lambda` is set to the largest allowed value. Parameters `scale` and `shape` are the maximum likelihood inference obtained from the full spike train (assuming it comes from a single neuron).

## Value

If model's initial guesses are not provided (with argument `initial.model`), the generated initial guesses are returned as a list (see [get.model.para](#)).

The following components are added or updated in list `spike.list`:

<code>generic.input.name</code>	The value of argument <code>generic.input.name</code> .
<code>start.time</code>	The value of argument <code>start.time</code> , if it did not exist before.
<code>stop.time</code>	The value of argument <code>stop.time</code> , if it did not exist before.
<code>energy</code>	The energy of the initial configuration and model parameters, if it did not exist before.
<code>cluster</code>	The vector cluster, if it did not exist before.
<code>state</code>	The vector state, if it did not exist before.

## Files written to disk

The following files are written to disk:

`generic.input.name.spike_train`: the data for the MCMC algorithm.

`generic.input.name.data_features`: some data features.

`generic.input.name.model_features`: some model features.

`generic.input.name.priors`: priors on model parameters.

`generic.input.name.neurons_1`: model parameters.

`generic.input.name.configuration_1`: spike train configuration.

`generic.input.name.nrj_1`: energy value.

## Author(s)

Christophe Pouzat and Matthieu Delescluse

## References

Christophe Pouzat, Matthieu Delescluse, Pascal Viot and Jean Diebolt (2004) Improved Spike-Sorting By Modeling Firing Statistics and Burst-Dependent Spike Amplitude Attenuation: A Markov Chain Monte Carlo Approach. *J Neurophys* **91**: 2910-2928.

Pouzat (2004) Technique(s) for Spike - Sorting. q-bio.QM/0405012. <http://fr.arxiv.org/abs/q-bio.QM/0405012>

## See Also

[get.model.para,reduce.and.whiten,display.neuron](#)

## Examples

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)
```

```
# Get intial guesses and write necessary files to disk before running
# the MCMC algorithm
initial.model <- pre.mcmc(spike.list = spike1, generic.input.name = "spike1", initial.model = NULL, nb
## End(Not run)
```

---

`read.configuration`     *Read a .configuration file*

---

## Description

This function is one of the functions used to analyse the output of the MCMC algorithm run by `mcmc`. Read a `.configuration` file that has been created by the Markov Chain Monte Carlo (MCMC) algorithm. Such files end with `"_beta"` where beta is the value of the inverse temperature at which the algorithm has been performed.

## Usage

```
read.configuration(spike.list = NULL, firstStep = NULL, beta = NULL)
```

## Arguments

<code>spike.list</code>	A list obtained with <code>display.energy.evolution</code> . If nothing is given an object browser will pop up.
<code>firstStep</code>	The first MCMC step to consider to compute the most probable configuration of the spike train. If nothing is given, a window will pop up to ask for it.
<code>beta</code>	The inverse temperature of the file to be read. If nothing is given, a window will pop up to ask for it.

## Details

The last argument `beta` specifies the suffix of the `.configuration` file to be read. `read.configuration` must be used after calling `display.energy.evolution` to check for convergence and choose the value of `firstStep`.

## Value

The function `read.configuration` does not return anything, it modifies its first argument by adding the following components:

```
cluster
state
```

: a one dimensional array with the "state" of each spike.

## Author(s)

Matthieu Delescluse

## References

Pouzat, Delescluse, Viot and Diebolt (2004) Improved Spike-Sorting By Modeling Firing Statistics and Burst-Dependent Spike Amplitude Attenuation: a Markov Chain Monte Carlo Approach. *J Neurophysiol* **91**: 2910-2928.

## See Also

[display.energy.evolution](#), [coderead.neurons](#), [get.mcmc.model.para](#), [display.events](#), [display.wilson](#)

## Examples

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)

# Write the input files of the MCMC algorithm
pre.model <- pre.mcmc(spike1)

# See the new components of spike1
str(spike1)

# Start up the MCMC algorithm
```

```

mcmc(spike1)

# See the new components of spike1
str(spike1)

# See the energy evolution
display.energy.evolution(spike1)

# See the new components of spike1
str(spike1)

# Read the output file .configuration
read.configuration(spike1)

# See the new components of spike1
str(spike1)

## End(Not run)

```

---

read.neurons	<i>Read a .neurons file</i>
--------------	-----------------------------

---

## Description

This function is one of the functions used to analyse the output of the MCMC algorithm run by `mcmc`. Read a `.neurons` file that has been created by the Markov Chain Monte Carlo (MCMC) algorithm. Such files end with `"_beta"` where beta is the value of the inverse temperature at which the algorithm has been performed.

## Usage

```
read.neurons(spike.list = NULL, beta = NULL)
```

## Arguments

<code>spike.list</code>	A list obtained with <code>mcmc</code> . If nothing is given an object browser will pop up. If nothing is given, a window will pop up to ask for it.
<code>beta</code>	The inverse temperature of the file to be read. If nothing is given, a window will pop up to ask for it.

## Details

The last argument `beta` specifies the suffix of the `.configuration` file to be read. `read.neurons` must be used after calling `display.energy.evolution` to check for convergence and choose the value of `firstStep`.

## Value

The function `read.neurons` does not return anything, it modifies its first argument by adding the following components:

```
neurons
```

: a matrix containing the value of each parameter (column) at each MCMC step (line). That is, this matrix has as many rows as there are MCMC steps and as many columns as there are model parameters. This matrix be analysed using the BOA package.

### Author(s)

Matthieu Delescluse

### References

Pouzat, Delescluse, Viot and Diebolt (2004) Improved Spike-Sorting By Modeling Firing Statistics and Burst-Dependent Spike Amplitude Attenuation: a Markov Chain Monte Carlo Approach. *J Neurophysiol* **91**: 2910-2928.

### See Also

[display.energy.evolution](#), [get.mcmc.model.para](#), [read.configuration](#), [display.events](#), [display.wilson](#)

### Examples

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)
```

```

# Write the input files of the MCMC algorithm
pre.model <- pre.mcmc(spike1)

# See the new components of spike1
str(spike1)

# Start up the MCMC algorithm
mcmc(spike1)

# See the new components of spike1
str(spike1)

# See the energy evolution
display.energy.evolution(spike1)

# See the new components of spike1
str(spike1)

# Read the output file .configuration
read.configuration(spike1)

# See the new components of spike1
str(spike1)

# Plot the events colored according to their label
display.events(spike1)

# Read the output file .neurons
read.neurons(spike1)

# See the new components of spike1
str(spike1)

## End(Not run)

```

---

reduce.and.whiten      *Reduce dimension of events space and whiten noise in this space*

---

## Description

Reduce the events space dimension by hand (*i.e.*, by clicking on an SD plot) or automatically by selecting a given number of local maxima of the sample SD, before performing noise whitening.

## Usage

```
reduce.and.whiten(spike.list = NULL, automatic = TRUE, nb.samples.per.site = 3)
```

## Arguments

**spike.list**      A list obtained with [make.sweeps](#). If nothing is given an object browser will pop up.

<code>automatic</code>	If TRUE the sample selection will be done automatically, if NOT, a graph will pop up and the user will have to select samples by clicking on them, if <code>automatic</code> is a vector its content will be used to specify the sample selection.
<code>nb.samples.per.site</code>	The number of samples per site to keep in the reduced space.

## Details

When the manual (that is, not automatic) samples selection is chosen, the user is expected to click on as many samples as given by the argument `nb.samples.per.site` on, say, the first recording site. The corresponding samples on the other sites will be automatically selected as well.

Once the selection is performed the spike sweeps and noise sweeps are reduced and the empirical noise covariance matrix is computed (using half of the noise sweeps). It is then use to whiten the (the other half of the) noise sweeps and the spike sweeps. The cumulative distribution of the whitened noise sweeps is then plotted together the expected distribution: a  $\chi^2$  distribution whose number of degrees of freedom is the product of `nb.samples.per.site` and of the number of recording sites (*i.e.*, the dimension of the reduced space). A more stringent Q-Q plot is also shown.

## Value

`reduce.and.whiten` function does not return anything but modifies its first argument. The following components are added to it:

<code>selected.indices</code>	An array with the indices of the selected samples.
<code>spike.sweeps.white</code>	The reduced and whitened spike sweeps.
<code>noise.mahalanobis</code>	An array with the empirical noise Mahalanobis distances of half of the (reduced) noise sweeps.

## Author(s)

Christophe Pouzat

## References

Pouzat, Mazor and Laurent (2002) Using noise signature to optimize spike-sorting and to assess neuronal classification quality. *J Neurosci Methods* **122**: 43-57.

Pouzat (2004) Technique(s) for Spike - Sorting. q-bio.QM/0405012. <http://fr.arxiv.org/abs/q-bio.QM/0405012>

## See Also

[make.sweeps](#), [display.wilson](#)

## Examples

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)

# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)

## End(Not run)
```

---

`select.data.files`      *function to select raw data files with a file browser and read them*

---

## Description

Calling `select.data.files` makes a file browser appear from which directories can be browsed and files selected, `read.data.files` reads selected files in R workspace.

## Usage

```
select.data.files()
read.data.files(trace.names = NULL, trace.length = 450000, size.of = NA)
```

**Arguments**

<code>trace.names</code>	A vector of character strings with the full path of the data to read
<code>trace.length</code>	The number of samples to read in each trace
<code>size.of</code>	The number of octets per data sample, default being the size of a double (8), for float give 4

**Details**

If no argument is given to `read.data.files`, a window will pop-up with a file browser inviting the user to select data files. If the data files have different length, the common length will be the shortest length.

**Value**

A character string vector containing the full paths of the selected files. A matrix with as many rows as data files (*i.e.*, the length of the first input argument) and as many columns as data samples.

**Author(s)**

Christophe Pouzat

**See Also**

[display.raw.data.gui](#)

**Examples**

```
## Not run:
# Start by selecting some data files
trace.names <- select.data.files()

# Notice that the data can be in a compressed format
# Now read the data which are (in that case) in double format
traces <- read.data.files(trace.names)

## End(Not run)
```

---

<code>spike.train.fwrite</code>	<i>Write to and read from disk spike train data in a format suitable for MCMC routines</i>
---------------------------------	--------------------------------------------------------------------------------------------

---

**Description**

Two routines to write and read spike train data before running the MCMC routines.

**Usage**

```
spike.train.fwrite(spike.list = NULL, file.name = NULL, nb.var = 4, sampling.frequency = 15)
spike.train.fread(file.name = NULL, nb.var = 4)
```

**Arguments**

<code>spike.list</code>	A list obtained with <code>reduce.and.whiten</code> . If nothing is given an object browser will pop up.
<code>file.name</code>	The name of the file where the spike train data will be written. If nothing is given a window will pop up inviting the user to select a folder and give a file name.
<code>nb.var</code>	The number of amplitude variables to use in the spike train.
<code>sampling.frequency</code>	The sampling frequency (in samples per second) used during data acquisition.
<code>start.time</code>	The time at which data collection was started. If nothing is given the largest integer smaller than the first spike time is selected.
<code>stop.time</code>	The time at which data collection ended. If nothing is given the smallest integer larger than the first spike time is selected.

**Details**

For `spike.train.fwrite`, if the whitened sweeps have more components than `nb.var`, then the `nb.var` components with the largest SD are selected.

**Value**

`spike.train.fread` returns a list with two components:

`spike.times` : a vector with the times of the individual spikes.

`spike.sweeps` : a matrix with as many rows as spikes and as many columns than `nb.var`.

`spike.train.fwrite` does not return anything

**Author(s)**

Christophe Pouzat

**References**

See description of the *Spike O Matic Library*.

**Examples**

```
## Not run:
# Select data files
trace.names <- select.data.files()

# Read data files
traces <- read.data.files(trace.names)

# If you feel like it (you should) look at them
display.raw.data.gui(traces)

# Detect spikes with template
spike1 <- find.spikes.with.template(traces,threshold=3)

# Look at the structure of spike1
str(spike1)
```

```
# Check out what the template looks like
plot(spike1$template)

# Check out the detection
display.detection.gui()

# Cut the sweeps
make.sweeps(spike1)

# See the new components of spike1
str(spike1)

# Look at the extracted sweeps
display.spike.sweeps(spike1)

# Look at the noise auto- and cross-correlation functions
display.noise.correlation(spike1)

# Reduce the sweeps length and whiten them
reduce.and.whiten(spike1)

# Write spike train data to disk
spike.train.fwrite(spike1, "spike1.spike_train", nb.var = 4)

# Read spike train
train1 <- spike.train.fread("spike1.spike_train", nb.var = 4)

## End(Not run)
```

# Index

- \*Topic **aplot**
  - display.raw.data.gui, 15
- \*Topic **cluster**
  - classify.from.mixture, 1
  - display.detection.gui, 3
  - display.energy.evolution, 4
  - display.energy.histo, 6
  - display.events, 8
  - display.model, 10
  - display.neuron, 12
  - display.noise.correlation, 14
  - display.spike.sweeps, 16
  - display.wilson, 18
  - find.spikes, 19
  - get.mcmc.model.para, 21
  - get.model, 28
  - get.model.para, 24
  - make.sweeps, 31
  - mcmc, 32
  - pre.mcmc, 35
  - read.configuration, 37
  - read.neurons, 40
  - reduce.and.whiten, 42
  - select.data.files, 44
  - spike.train.fwrite, 45
- \*Topic **iplot**
  - display.raw.data.gui, 15
- classify.from.mixture, 1, 10–12, 24, 25, 27, 35
- cor.test, 25
- display.detection.gui, 3, 16, 20
- display.energy.evolution, 4, 4, 6, 8, 9, 21, 22, 38, 41
- display.energy.histo, 6
- display.events, 5, 8, 8, 22, 38, 41
- display.model, 10
- display.neuron, 12, 17, 37
- display.noise.correlation, 14, 18, 32
- display.raw.data.gui, 15, 45
- display.spike.sweeps, 12, 16, 18, 32
- display.wilson, 5, 9, 18, 22, 30, 32, 38, 41, 43
- find.spikes, 19
- find.spikes.no.template, 3, 31, 32
- find.spikes.no.template  
(*find.spikes*), 19
- find.spikes.with.template, 3, 31, 32
- find.spikes.with.template  
(*find.spikes*), 19
- get.mcmc.model.para, 5, 9, 21, 38, 41
- get.model, 1, 11, 24, 25, 27, 28
- get.model.para, 10, 11, 21, 24, 35–37
- make.noise.sweeps, 14, 17, 20
- make.noise.sweeps (*make.sweeps*), 31
- make.spike.sweeps, 14, 16–18, 20
- make.spike.sweeps (*make.sweeps*), 31
- make.sweeps, 14, 16–18, 20, 31, 42, 43
- mcmc, 4, 6, 8, 21, 32, 38, 40
- nls, 25
- pre.mcmc, 33, 34, 35
- read.configuration, 4, 5, 8, 9, 22, 37, 41
- read.data.files, 16
- read.data.files (*select.data.files*), 44
- read.neurons, 4, 5, 9, 21, 22, 38, 40
- reduce.and.whiten, 1, 12, 18, 28, 30, 32, 35, 37, 42, 45
- select.data.files, 16, 44
- spike.train.fread  
(*spike.train.fwrite*), 45
- spike.train.fwrite, 45