Package ‘scbursts’

October 14, 2022

Version 1.6
Date 2019-06-24
Title Single Channel Bursts Analysis
Description Provides tools to import and export from several existing pieces of ion-channel analysis software such as ‘TAC’, ‘QUB’, ‘SCAN’, and ‘Clampfit’, implements procedures such as dwell-time correction and defining bursts with a critical time, and provides tools for analysis of bursts, such as tools for sorting and plotting.
Author Blair Drummmond [aut], Mathieu Dextraze [ctb]
Maintainer Blair Drummond <blair.robert.drummond@gmail.com>
License LGPL-2.1
LazyData true
RoxygenNote 6.1.1
Suggests knitr, rmarkdown, tinytex
VignetteBuilder knitr
Imports readxl, tibble
NeedsCompilation no
Repository CRAN
Date/Publication 2019-07-06 15:30:03 UTC

R topics documented:

bursts.check_subconductance ........................................... 3
bursts.conductance_states ............................................. 3
bursts.copy ............................................................ 4
bursts.defined_by_tcrit ................................................ 4
bursts.get_gaps .......................................................... 5
bursts.impose_deadtime .................................................. 6
bursts.modify_conductance .............................................. 7
bursts.pcloseds .......................................................... 8
bursts.popens ........................................................... 8
bursts.recombine ....................................................... 9
bursts.remove_first_and_last ........................................ 10
bursts.select .......................................................... 11
bursts.sort ............................................................... 12
bursts.space_out ......................................................... 13
bursts.start_times_update ............................................... 14
bursts.subconductance_as ............................................ 14
clampfit.read .......................................................... 15
cplot.conductance_hist ................................................ 16
cplot.log_root_axes .................................................... 16
cplot.pclosed_ts ....................................................... 17
dwt.read ................................................................. 18
dwt.write ............................................................... 19
evt.extract_header ..................................................... 20
evt.from_dwells .......................................................... 20
evt.read ................................................................. 21
evt.to_dwells ............................................................. 22
evt.write ............................................................... 22
hst.extract_header ...................................................... 23
hst.read ................................................................. 24
hst.write ............................................................... 24
risetime.correct_gaussian ............................................. 25
scan.read ............................................................... 26
segment.check_subconductance ...................................... 27
segment.closed_dwells ................................................ 27
segment.conductance_states ......................................... 28
segment.consecutives_to_dwells .................................... 29
segment.copy ........................................................... 29
segment.count_closed ................................................ 30
segment.count_dwells ................................................ 30
segment.count_open ................................................... 31
segment.create ........................................................ 32
segment.duration ....................................................... 33
segment.dwells_by_conductance ................................... 33
segment.dwells_by_conductance_range ............................. 34
segment.impose_deadtime ............................................. 35
segment.modify_conductance ........................................ 35
segment.name .......................................................... 36
segment.open_dwells .................................................. 37
segment.pclosed ....................................................... 38
segment.pconductance ............................................... 38
segment.pconductance_range ....................................... 39
segment.popen .......................................................... 40
segment.sseg ........................................................... 40
segment.start_time .................................................... 41
segment.subconductance_as ......................................... 42
segment.verify ........................................................ 42
util.basename ........................................................... 43
bursts.check_subconductance

Check if segment contains subconductive states

Description

Check if segment contains subconductive states

Usage

bursts.check_subconductance(bursts)

Arguments

bursts The list of all bursts

Value

True if it contains an conductance other than 0 or 1, False otherwise.

Examples

infile <- system.file("extdata", "example4.dwt", package = "scbursts")
dwells <- dwt.read(infile)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")
bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")
bursts.check_subconductance(bursts)

bursts.conductance_states

Return a list of all the (sub)conductance states.

Description

Return a list of all the (sub)conductance states.

Usage

bursts.conductance_states(bursts)

Arguments

bursts The list of all bursts
bursts.defined_by_tcrit

Value

a list of all the (sub)conductance states.

Examples

```r
infile <- system.file("extdata", "example4.dwt", package = "scbursts")
dwells <- dwt.read(infile)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")
bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")

bursts.conductance_states(bursts)
```

---

bursts.copy

*Copy a list of bursts (by value)*

Description

Copy a list of bursts (by value)

Usage

```r
bursts.copy(bursts)
```

Arguments

- `bursts`: bursts to copy

Value

A copy of the bursts.

---

bursts.defined_by_tcrit

*Divide a recording into bursts defined by a critical time.*

Description

Split segment at long pauses, dividing the segment into multiple shorter segments (which are the bursts). Along with the interburst closings, which are referred to as "gaps". (Default time units are seconds)

Usage

```r
bursts.defined_by_tcrit(segments, t_crit, units = "s")
```
bursts.get_gaps

Arguments

segments A segment or multiple segments with $states and $dwells. NOTE: separate segments will remain split, regardless of why they were originally divided.
t_crit Critical time at which to divide bursts (in seconds by default)
units what unit the critical time is in ("s", "ms", "us", or "ns")

Value

bursts. Which is a list of segments starting and ending in 1 states (open dwell)

Examples

infile <- system.file("extdata", "example1_tac.evt", package = "scbursts")
transitions <- evt.read(infile)
dwells <- evt.to_dwells(transitions)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")

bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")
head(bursts[[1]])

----------------------------------------
burts.get_gaps Get the gaps between bursts.
----------------------------------------

Description

Extract vector of gaps from the bursts. This is done using the start_time attribute, which is mostly hidden in the data. (The gaps at the ends may have length 0)

Usage

bursts.get_gaps(bursts)

Arguments

bursts The list of segments

Value

A vector of N+1 gaps for N bursts times
Examples

```r
infile <- system.file("extdata", "example1_tac.evt", package = "scbursts")
transitions <- evt.read(infile)
dwells <- evt.to_dwells(transitions)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")

bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")
gaps <- bursts.get_gaps(bursts)

head(gaps)
```

---

**bursts.impose_deadtime**

*Imposes a deadline to each segment in a burst.*

Description

The user specifies a deadline in microseconds. The function applies segment.impose_deadtime to each segment in the burst. (See segment.impose_deadtime for details.)

Usage

```r
bursts.impose_deadtime(bursts, deadtime)
```

Arguments

- **bursts**: a burst containing segments of dwells and states.
- **deadtime**: the briefest possible event in microseconds.

Value

A modified copy of the original burst

Examples

```r
infile <- system.file("extdata", "example4.dwt", package = "scbursts")
dwells <- dwt.read(infile)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")
bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")

bursts_d <- bursts.impose_deadtime(bursts, deadtime=0.01)
```
bursts.modify_conductance

Transform the conductance states according to a user-defined function of conductance level.

Description

Transform the conductance states according to a user-defined function of conductance level.

Usage

bursts.modify_conductance(bursts, fun)

Arguments

- **bursts**: the list of segments
- **fun**: a function on conductance levels

Value

A modified copy of the original bursts

Examples

```r
infile <- system.file("extdata", "example4.dwt", package = "scbursts")
dwells <- dwt.read(infile)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")
bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")

### Collapse into three subconductance states
fun <- function(amp) {
  if (amp < 0.3)
    return(0)
  else if (amp >= 0.3 && amp < 0.6)
    return(0.5)
  else
    return(1)
}

bursts_d <- bursts.modify_conductance(bursts, fun)
```
bursts.popens

---

**bursts.popens** | Return popens of every burst.

---

**Description**

Return popens of every burst.

**Usage**

`bursts.popens(bursts)`

**Arguments**

- **bursts** | The list of all bursts

**Value**

The pclosed values

**Examples**

```r
infile <- system.file("extdata", "example1_qub.dwt", package = "scbursts")
dwells <- dwt.read(infile)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")
bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")

pcloseds <- bursts.popens(bursts)
hist(pcloseds)
```
Value

The popen values

Examples

```r
infile <- system.file("extdata", "example1_qub.dwt", package = "scbursts")
dwells <- dwt.read(infile)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")

bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")
popens <- bursts.popens(bursts)
hist(popens)
```

```
bursts.recombine Combine bursts into one recording (with obvious spaces between them).
```

Description

From a list of segments, return the concatenated segment containing all bursts. Inverse of functions like bursts.defined_by_tcrit

Usage

`bursts.recombine(bursts)`

Arguments

- `bursts` The list of all bursts

Value

The segment containing all bursts.

Examples

```r
infile <- system.file("extdata", "example1_qub.dwt", package = "scbursts")
dwells <- dwt.read(infile)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")

bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")

# This is a single segment!
record <- bursts.recombine(bursts)
```
# Which means you can do stuff like this
open_dwells <- segment.open_dwells(bursts.recombine(bursts))

 bursts.remove_first_and_last

Remove the first and last burst from the list.

Description

Remove the first and last burst from the list.

Usage

bursts.remove_first_and_last(bursts)

Arguments

bursts  The list of all bursts

Value

A shorter list of bursts

Examples

infile <- system.file("extdata", "example1_tac.evt", package = "scbursts")
transitions <- evt.read(infile)
dwells <- evt.to_dwells(transitions)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")

bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")

# If there seem to be bad bursts at the ends
bursts <- bursts.remove_first_and_last(bursts)
bursts.select

From a list of bursts, extract those that interest you by passing a selecting function.

Description

From a list of bursts, extract those that interest you by passing a selecting function.

Usage

bursts.select(bursts, func, one_file = FALSE)

Arguments

- **bursts**: The list of all bursts
- **func**: A function of a segment that returns either TRUE or FALSE
- **one_file**: TRUE or FALSE: Return a single file to write to disk, or a list of bursts. The one_file will return a file with all unselected bursts zeroed out.

Value

A shorter list of bursts OR if one_file is passed one segment with zeros where the other bursts might have been originally. Defaults to FALSE.

Examples

```r
high_popen <- function (seg) {
  segment.popen(seg) > 0.7
}

infile <- system.file("extdata", "example1_qub.dwt", package = "scbursts")
dwells <- dwt.read(infile)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")

bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")
subset <- bursts.select(bursts, high_popen)

# To export to one .dwt file
subset_f <- bursts.select(bursts, high_popen, one_file=TRUE)
```
bursts.sort  
Order a list of bursts by some function. For instance, `popen`.

Description
Order a list of bursts by some function. For instance, `popen`.

Usage
```
bursts.sort(bursts, func, reverse = FALSE)
```

Arguments
- **bursts**: The list of all bursts
- **func**: A function of a segment that returns a numeric value
- **reverse**: By default, return in ascending order. Use `reverse=TRUE` to change that.

Value
A list sorted by `func`. By default in ascending order (unless reversed)

Examples
```
# Wittenberg, Bursts and Bursting Patterns

infile <- system.file("extdata", "example1_qub.dwt", package = "scbursts")
dwells <- dwt.read(infile)
dwells.c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")
bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")

# A sorted list of bursts.
sorted <- bursts.sort(bursts, segment.popen)

# You can also write your own functions. If you want `P(Open) = P(Closed)`
variance_fun <- function (seg) {
  # Any function that maps a segment to a number works.
  return(  segment.popen(seg) * segment.pclosed(seg)  )
}

weird_sort <- bursts.sort(bursts, variance_fun)
```
bursts.space_out

Artificially add amount of time between bursts (in absence of recording information).

**Description**

Given a list of segments separated by an unknown amount of time, one may want to space the segments by some amount of time, so that they can be plotted. This function takes a separating factor, and splits up the segments by either that factor (in seconds), or that many multiples of the largest observed dwell.

**Usage**

```
bursts.space_out(segments, sep_factor = 1000)
```

**Arguments**

- `segments`: The segments to space out
- `sep_factor`: the factor by which to separate the segments. Either the factor in seconds, or a multiple of the longest observed dwell.

**Value**

The segments again, but with modified meta-data.

**Examples**

```
infile <- system.file("extdata", "example_multiple_segments.dwt", package = "scbursts")
dwells <- dwt.read(infile)

# Still a list, but the meta-data is fixed
spaced_records <- bursts.space_out(dwells, sep_factor=1000)

# Combine them, and they'll be nicely spaced out.
single_record <- bursts.recombine(spaced_records)

# You can now plot that single_record using one of the plot functions.
```
bursts.start_times_update

(DON’T USE THIS) Fix meta-data of bursts.

Description

YOU PROBABLY WON’T EVER HAVE TO CALL THIS DIRECTLY. Attach the meta-data to each segment saying when it began. It interleaves the durations of the bursts and gaps, and assigns the sum of those durations up to a point as the starting time.

Usage

bursts.start_times_update(bursts, gaps)

Arguments

bursts List of segments

gaps vector of gap times.

Value

A list of segments, one per burst, with updated start_times

bursts.subconductance_as

Imposes a fixed conductance level (0 or 1) to all dwells with subconductance levels to each segment in a burst

Description

The user specifies the desired level (’open’ or ’closed’). The function applies segment.subconductance_as to each segment in the burst. (See segment.subconductance_as for details.)

Usage

bursts.subconductance_as(bursts, level)

Arguments

bursts the list of segments

level either 'open' or 'closed'

Value

A modified copy of the original burst
**clampfit.read**

Read a .xlsx file output from clampfit

---

**Description**

Read a .xlsx file output from clampfit. Result is a list of "segments", which is a dataframe extra data. See "segment" for more details. Converts millisecond dwells to seconds.

**Usage**

```r
clampfit.read(filename, separating_factor = 1000, header = FALSE)
```

**Arguments**

- `filename` Filenname to read from
- `separating_factor` In lieu of a known time between segments, separate with a multiple of the longest dwell.
- `header` Does the file include a header?

**Value**

A list of bursts (possibly a singleton)

**Examples**

```r
infile <- system.file("extdata", "example1_clipfit.xlsx", package = "scbursts")
dwells <- clampfit.read(infile)
head(dwells)
```
### cplot.conductance_hist

*Histogram of Conductance States*

**Description**

Histogram of Conductance States

**Usage**

```r
cplot.conductance_hist(bursts, ...)```

**Arguments**

- **bursts**: List of multiple segments
- **...**: other arguments passed to histogram

**Examples**

```r
infile <- system.file("extdata", "example4.dwt", package = "scbursts")
dwells <- dwt.read(infile)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")
bursts <- bursts.defined_by_tcrite(dwells_c, 100, units="ms")
cplot.conductance_hist(bursts, main="example4.dwt conductance state histogram")
```

### cplot.log_root_axes

*Add log-root axes to histogram plot*

**Description**

Add log-root axes to histogram plot

**Usage**

```r
cplot.log_root_axes(points)```

**Arguments**

- **points**: The data to plot
**cplot.pclosed_ts**

*Plot Time Series (ts) of P(Closed).*

**Description**

Plot Time Series (ts) of P(Closed).

**Usage**

```r
cplot.pclosed_ts(bursts, main = "P(Closed) Time Series", ...)
```

**Arguments**

- `bursts` List of multiple segments
- `main` The title of the plot.
- `...` Options to pass to plot

**Examples**

```r
infile <- system.file("extdata", "example1_qub.dwt", package = "scbursts")
dwells <- dwt.read(infile)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")
bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")

open_dwells <- segment.open_dwells(bursts.recombine(bursts))
hist(log10(open_dwells), axes=FALSE, breaks=30)
cplot.log_root_axes(open_dwells)
```

```r
infile <- system.file("extdata", "example1_qub.dwt", package = "scbursts")
dwells <- dwt.read(infile)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")
bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")

cplot.pclosed_ts(bursts, main="P(Closed) Time Series, 2018-09-20")
```
cplot.popen_ts  
*Plot Time Series (ts) of P(Open).*

**Description**

Plot Time Series (ts) of P(Open).

**Usage**

cplot.popen_ts(bursts, main = "P(Open) Time Series", ...)

**Arguments**

- **bursts**: List of multiple segments
- **main**: The title of the plot.
- **...**: Options to pass to plot

**Examples**

```r
infile <- system.file("extdata", "example1_qub.dwt", package = "scbursts")
dwells <- dwt.read(infile)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")
bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")

cplot.popen_ts(bursts, "P(Open) Time Series, 2018-09-20")
```

dwt.read  
*Read a .dwt file.*

**Description**

Read a .dwt file. Result is a list of "segments", which is a dataframe extra data. See "segment" for more details. Converts millisecond dwells to seconds.

**Usage**

dwt.read(filename, separating_factor = 1000)

**Arguments**

- **filename**: Filename to read from
- **separating_factor**: In lieu of a known time between segments, separate with a multiple of the longest dwell.
dwt.write

Value

A list of bursts (possibly a singleton)

Examples

```r
infile <- system.file("extdata", "example1_tac.evt", package = "scbursts")
transitions <- evt.read(infile)
dwells <- evt.to_dwells(transitions)

dwt.write(dwells, file=file.path(tempdir(), "dwells.dwt"))

# Quit R, come back the next day
## Not run:
dwells <- dwt.read("dwells.dwt")
## End(Not run)
```

---

dwt.write  
Write a dwt file to disk. Writes DOS line endings. Dwells are in milliseconds

Description

Write a dwt file to disk. Writes DOS line endings. Dwells are in milliseconds

Usage

dwt.write(segments, file = "", seg = 1, append = FALSE)

Arguments

- **segments**: A segment or multiple segments with $dwells and $states
- **file**: Filename to write to
- **seg**: Segment number to write in .dwt header.
- **append**: Add to the end of a file or overwrite? (defaults to false)

Examples

```r
infile <- system.file("extdata", "example1_tac.evt", package = "scbursts")
transitions <- evt.read(infile)
dwells <- evt.to_dwells(transitions)

dwt.write(dwells, file=file.path(tempdir(), "dwells.dwt"))
```
**evt.extract_header**  
*Extract header from evt file.*

**Description**
Extract header from evt file.

**Usage**
```
evt.extract_header(filename)
```

**Arguments**
- **filename**  
The filename

**Value**
A string containing the header

**Examples**
```
infile <- system.file("extdata", "example1_tac.evt", package = "scbursts")

# Get Dwell
transitions <- evt.read(infile)
dwells <- evt.to_dwells(transitions)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")

# Get Header
header <- evt.extract_header(infile)

evt.write(dwells_c, header=header, file=file.path(tempdir(), "fixed_example1_tac.evt"))
```

**evt.from_dwells**  
*Converts dwell durations to absolute transition times.*

**Description**
Converts dwell durations to absolute transition times.

**Usage**
```
evt.from_dwells(segments)
```

**Arguments**
- **segments**  
A segment or multiple segments
**evt.read**

Value

A dataframe or multiple dataframes of states and transition times

Examples

```r
dwells_file <- system.file("extdata", "example1_qub.dwt", package = "scbursts")
dwells <- dwt.read(dwells_file)
transitions <- evt.from_dwells(dwells)
```

Description

Read a .evt file to a table. Times are in seconds

Usage

```r
evt.read(filename)
```

Arguments

```r
filename
```

The filename

Value

A list of tables with columns "states" and "times". Each table corresponds to a contiguous segment from a recording.

Examples

```r
# import some of the data included with the package
infile <- system.file("extdata", "example1_tac.evt", package = "scbursts")
transitions <- evt.read(infile)
head(transitions[[1]])
```
**evt.to_dwells**

*Calculate pulse lengths. Converts transition times to dwell durations.*

**Description**

Calculate pulse lengths. Converts transition times to dwell durations.

**Usage**

```r
evt.to_dwells(tables)
```

**Arguments**

- `tables`: Either a single table or a list of tables with columns "states" and "times".

**Value**

A segment or a list of segments with one less row, where each row represents pulse in state 0 (closed dwell) of duration 0.51231, instead of the time at which the state transitioned.

**Examples**

```r
infile <- system.file("extdata", "example1_tac.evt", package = "scbursts")
transitions <- evt.read(infile)
dwells <- evt.to_dwells(transitions)
head(dwells[[1]])
```

---

**evt.write**

*Write bursts to a .evt file.*

**Description**

Write bursts to a .evt file.

**Usage**

```r
evt.write(segments, filename = "", header = NULL)
```

**Arguments**

- `segments`: A segment or list of segments to write to filename.
- `filename`: The filename.
- `header`: The header information for the .evt file, if available.
Examples

```r
infile <- system.file("extdata", "example1_tac.evt", package = "scbursts")

# Get Dwell
transitions <- evt.read(infile)
dwells <- evt.to_dwells(transitions)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")

# Get Header
header <- evt.extract_header(infile)

evt.write(dwells_c, header=header, file=file.path(tempdir(), "fixed_example1_tac.evt"))
```

**hst.extract_header**

Extract header from hst file.

**Description**

Extract header from hst file.

**Usage**

```r
hst.extract_header(filename)
```

**Arguments**

- **filename** The filename

**Value**

A string containing the header

**Examples**

```r
# import some of the data included with the package
infile <- system.file("extdata", "example1_hst.hst", package = "scbursts")

open_table <- hst.read(infile, extract="open")
closed_table <- hst.read(infile, extract="closed")
header <- hst.extract_header(infile)

# Make adjustments to the histogram, if you wish
hst.write(open_table, closed_table, file=file.path(tempdir(), "output_hist.hst"), header=header)
```
hst.read

Read a MIL ".hst" file to a table.

Description

Read a MIL ".hst" file to a table. By default these files are in log10(Milliseconds)-sqrt(Freq), but unless "raw" is set to TRUE, this function returns a table containing Seconds-Freq

Usage

hst.read(filename, extract = "open", raw = FALSE)

Arguments

filename
  The filename
extract
  Extract either "open" or "closed" histogram
raw
  Data is given as log10(milliseconds)-Sqrt(Freq). Setting raw=FALSE yields output as Seconds-Frequency

Value

A tables with columns "bin", "freq" and "fit".

Examples

  # import some of the data included with the package
  infile <- system.file("extdata", "example1_hst.hst", package = "scbursts")
  open_hst <- hst.read(infile, extract="open")
  closed_hst <- hst.read(infile, extract="closed")

  head(open_hst)
  head(closed_hst)

hst.write

Write bursts to a log10(ms)-sqrt(Freq) .hst file from open and closed tables.

Description

Write bursts to a log10(ms)-sqrt(Freq) .hst file from open and closed tables.

Usage

hst.write(open_hist, closed_hist, file = "", header = NULL,
  fromraw = FALSE)
**Arguments**

- `open_hist`: The table (bin,freq,fit) for open times
- `closed_hist`: The table (bin,freq,fit) for closed times
- `file`: The filename
- `header`: The header info
- `fromraw`: Unless FALSE, assume we need to write a log10(milliseconds)-sqrt(Frequency) plot

**Examples**

```r
infile <- system.file("extdata", "example1_hst.hst", package = "scbursts")
open = hst.read(infile, extract="open")
closed = hst.read(infile, extract="closed")
header = hst.extract_header(infile)

### Do stuff
hst.write(open, closed, file=file.path(tempdir(), "new_histogram.hst"), header=header)
```

**Description**

Undo the effect of the gaussian filter. See section 4.1.1 of Colquhoun and Sigworth, "Fitting and Analysis of Single-Channel segments". NOTE: This is potentially problematic, in that this unfiltering lengthens every dwell. A less naive algorithm would take into account the influence of the surroundings, as they impact the effects of the filter.

**Usage**

```r
risetime.correct_gaussian(Tr, segments, units = "s")
```

**Arguments**

- `Tr`: Rise time of the filter in (us)
- `segments`: A segment or multiple segments with $states and $dells to correct.
- `units`: What unit the risetime is input in (defaults to seconds)

**Value**

A Segment or multiple segments with corrected risetimes.
Examples

```r
infile <- system.file("extdata", "example1_tac.evt", package = "scbursts")
transitions <- evt.read(infile)
dwells <- evt.to_dwells(transitions)

dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")
```

Description

Data is in seconds.

Usage

```r
scan.read(filename, separating_factor = 1000)
```

Arguments

- `filename`, the file name to read from.
- `separating_factor` In lieu of a known time between segments, separate with a multiple of the longest dwell.

Value

A list of recording segments from the scan file

Examples

```r
infile <- system.file("extdata", "example1_scan.txt", package = "scbursts")
record <- scan.read(infile)
head(record)
```
**segment.check_subconductance**

*Check if segment contains subconductive states*

**Description**

Check if segment contains subconductive states

**Usage**

```
segment.check_subconductance(segment)
```

**Arguments**

- `segment` The dwells and states table

**Value**

True if it contains an conductance other than 0 or 1, False otherwise.

**Examples**

```
# It's more likely that you created states or dwells with some function
states <- c(0, 0.2, 0, 1, 0, 0.5, 0, 0.7, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")

segment.check_subconductance(my_burst)
```

---

**segment.closed_dwells**  
*Extract closed dwells.*

**Description**

Extract closed dwells.

**Usage**

```
segment.closed_dwells(segment)
```

**Arguments**

- `segment` the segment object
segment.conductance_states

Value

the closed dwells

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 1, 0, 1, 0, 1, 0, 1, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")
closed_dwells <- segment.closed_dwells(my_burst)
head(closed_dwells)

segment.conductance_states

Return a list of all the (sub)conductance states.

Description

Return a list of all the (sub)conductance states.

Usage

segment.conductance_states(segment)

Arguments

segment The dwells and states table

Value

a list of all the (sub)conductance states.

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 0.2, 0, 1, 0, 0.5, 0, 0.7, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")
segment.conductance_states(my_burst)
**segment.consecutives_to_dwells**

*Collapses a segment into dwells with alternating conductance levels.*

**Description**

Segments may contain consecutive dwells with the same conductance level. `consecutives_to_dwells` sums together all consecutive dwells with the same conductance level. The result is a segment containing dwells that alternate in conductance level (i.e. 1,0,1,0,1,...)

**Usage**

`segment.consecutives_to_dwells(segment)`

**Arguments**

- `segment` The dwells and states table

**Value**

A modified copy of the original segment

---

**segment.copy**

*Copy a segment*

**Description**

This is a low-level function, mostly for use internally by other functions. There aren’t many reasons to use this.

**Usage**

`segment.copy(segment)`

**Arguments**

- `segment` The segment to copy

**Value**

A duplicate identical content.
segment.count_closed  
*Extract number of closed dwells. In the case of subconductive states, a dwell is only closed if the conductance is exactly zero.*

**Description**

Extract number of closed dwells. In the case of subconductive states, a dwell is only closed if the conductance is exactly zero.

**Usage**

```r
segment.count_closed(segment)
```

**Arguments**

- `segment`: the segment object

**Value**

number of closed dwells

**Examples**

```r
# It's more likely that you created states or dwells with some function
states <- c(0, 1, 0, 1, 0, 1, 0, 1, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")

segment.count_closed(my_burst)
```

---

segment.count_dwells  
*Extract number of dwells in segment.*

**Description**

Extract number of dwells in segment.

**Usage**

```r
segment.count_dwells(segment)
```

**Arguments**

- `segment`: the segment object
Value

number of dwells

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 1, 0, 1, 0, 1, 0, 1, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")
segment.count_open(my_burst)
segment.create

Create a "segment" object

Description

This is a low-level function, mostly for use internally by other functions. There aren't many reasons to use this. Create object containing table data and metadata. The object can be used as a dataframe, and the metadata can be accessed with the functions: segment.seg, segment.start_time, segment.filename

Usage

segment.create(states, dwells, seg = 1, start_time = 0,
               name = "burst", ignore_errors = FALSE)

Arguments

states  a vector of states
dwells  a vector of dwell durations (same length as states)
seg     The segment number. Defaults to 1
start_time  When the dwells began. Defaults to 0
name    Suffix-less version of the original filename. 60uM.dwt -> '60uM'
ignore_errors  Do not report faulty segments (not many reasons to do this)

Value

The segment object: A dataframe with extra metadata.

Examples

# It's more likely that you created states or dwells with some function
states <- c(0,  1, 0, 1, 0, 1, 0, 1, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=0, name="example_segment")

segment.name(my_burst)
segment.duration

\( \text{Get duration of a segment.} \)

**Description**

Get duration of a segment.

**Usage**

\[
\text{segment.duration}(\text{segment})
\]

**Arguments**

- **segment**: the segment object

**Value**

the duration

**Examples**

```r
# It's more likely that you created states or dwells with some function
dstates <- c(0, 1, 0, 1, 0, 1, 0, 1, 0, 1)
ddwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(dstates, ddwells, seg=1, start_time=3.14159, name="example_segment")

segment.duration(my_burst)
```

---

**segment.dwells_by_conductance**

\( \text{Extract dwells in conductance range. lower} \leq x \leq \text{upper} \)

**Description**

Extract dwells in conductance range. lower \( \leq x \leq \) upper

**Usage**

\[
\text{segment.dwells_by_conductance}(\text{segment, level})
\]

**Arguments**

- **segment**: the segment object
- **level**: The conductance to extract
segment.dwells_by_conductance_range

Value

the dwells in a given range

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 0.2, 0, 1, 0, 0.5, 0, 0.7, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")

half_open <- segment.dwells_by_conductance(my_burst, lower=0.2, upper=0.7)
head(half_open)
segment.impose_deadtime

Imposes a deadtime to a segment by removing any dwell that is shorter than the deadtime.

Description

The user specifies a deadtime in microseconds. The function effectively undoes the work of the event detection algorithm by reverting the conductance level (of the brief dwell) back to the previous conductance level in the time sequence. The function then returns a collapsed segment containing alternating dwells.

Usage

segment.impose_deadtime(segment, deadtime)

Arguments

segment the segment containing dwells and states.
deadtime the briefest possible event in microseconds.

Value

A modified copy of the original segment

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 0.2, 0, 1, 0, 0.5, 0, 0.7, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")

my_burst_d <- segment.impose_deadtime(my_burst, deadtime=0.3)

segment.modify_conductance

Transform the conductance states according to a user-defined function of conductance level.

Description

Transform the conductance states according to a user-defined function of conductance level.
Usage

segment.modify_conductance(segment, fun)

Arguments

segment the segment containing dwells and states.
fun a function on conductance levels (states)

Value

A modified copy of the original segment

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 0.2, 0, 1, 0, 0.5, 0, 0.7, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")

### Collapse into three subconductance states
fun <- function(amp) {
  if (amp < 0.3)
    return(0)
  else if (amp >= 0.3 && amp < 0.6)
    return(0.5)
  else
    return(1)
}

my_burst_d <- segment.modify_conductance(my_burst, fun)

segment.name

Extract name from segment.

Description

Extract name from segment.

Usage

segment.name(segment)

Arguments

segment the segment object
Value

Segment name (string)

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 1, 0, 1, 0, 1, 0, 1, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")

segment.name(my_burst)

segment.open_dwells

Extract open dwells. (Any conductance greater than zero)

Description

Extract open dwells. (Any conductance greater than zero)

Usage

segment.open_dwells(segment)

Arguments

segment the segment object

Value

the open dwells

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 1, 0, 1, 0, 1, 0, 1, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")

open_dwells <- segment.open_dwells(my_burst)
head(open_dwells)
segment.pclosed

Calculate empirical P(Closed) of a segment.

Description

Calculate empirical P(Closed) of a segment. NOTE: Assuming that burst starts and ends with 1

Usage

segment.pclosed(segment)

Arguments

segment
The dwells and states table

Value

The ratio of closed time to total time

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 1, 0, 1, 0, 1, 0, 1, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")

# P(Closed) of this burst
segment.pclosed(my_burst)

segment.pconductance

Calculate empirical P(Lower <= Conductance <= Upper) of a segment.

Description

Calculate empirical P(Lower <= Conductance <= Upper) of a segment.

Usage

segment.pconductance(segment, level)

Arguments

segment
the segment object
level
conductance level
segment.pconductance_range

Value

The probability of being in this conductance state

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 0.2, 0, 1, 0, 0.5, 0, 0.7, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")

segment.pconductance(my_burst, 0.5)

segment.pconductance_range

Calculate empirical \( P(\text{Lower} \leq \text{Conductance} \leq \text{Upper}) \) of a segment.

Description

Calculate empirical \( P(\text{Lower} \leq \text{Conductance} \leq \text{Upper}) \) of a segment.

Usage

segment.pconductance_range(segment, lower = 0, upper = Inf)

Arguments

segment the segment object
lower lower bound on conductance (defaults to 0)
upper upper bound on conductance (defaults to infinity)

Value

The probability of being in these conductance states

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 0.2, 0, 1, 0, 0.5, 0, 0.7, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")

segment.pconductance_range(my_burst, lower=0.5, upper=0.5)
### segment.popen

*Calculate empirical P(Open) of a segment. (A state is considered open if the conductance is non-zero)*

**Description**

Calculate empirical P(Open) of a segment. NOTE: Assuming that burst starts and ends with 1

**Usage**

```
segment.popen(segment)
```

**Arguments**

- `segment` - The dwells and states table

**Value**

The ratio of open time to total time

**Examples**

```r
# It's more likely that you created states or dwells with some function
states <- c(0, 1, 0, 1, 0, 1, 0, 1, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")

# P(Open) of this burst
segment.popen(my_burst)
```

### segment.seg

*Extract segment number from segment.*

**Description**

Extract segment number from segment.

**Usage**

```
segment.seg(segment)
```

**Arguments**

- `segment` - the segment object
Value

Segment number (integer)

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 1, 0, 1, 0, 1, 0, 1, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=0, name="example_segment")

segment.start_time(my_burst)

---

**segment.start_time**  
Extract start_time from segment.

Description

Extract start_time from segment.

Usage

segment.start_time(segment)

Arguments

segment the segment object

Value

Segment start_time (float)

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 1, 0, 1, 0, 1, 0, 1, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")

segment.start_time(my_burst)
segment.subconductance_as

Imposes a fixed conductance level (0 or 1) to all dwells with subconductance levels.

Description

The user specifies the desired level ('open' or 'closed'). The function will modify any subconductance level (that is not 0 or 1) to be the desired level 1 for 'open' or 0 for 'closed'. The function then returns a collapsed segment containing alternating dwells. (See segment.consecutives_to_dwells for details about the collapsed segment.)

Usage

segment.subconductance_as(segment, level)

Arguments

segment the segment containing dwells and states.
level either 'open' or 'closed'

Value

A modified copy of the original segment

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 0.2, 0, 1, 0, 0.5, 0, 0.7, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")

my_burst_d <- segment.subconductance_as(my_burst, "open")

segment.verify

Detect misrecorded data.

Description

Segments should have a very specific shape, but recordings can produce errors that make nonsensical segments. In particular, ones contain multiple consecutive states of equal conductance, or end in closings. This function detects whether a segment satisfies the constraint that the segment conductances are not the same from one dwell to the next, and begin and end with a closing.
util.basename

Usage

segment.verify(segment)

Arguments

segment The dwells and states table

Value

True if a valid segment, False otherwise

Examples

# It's more likely that you created states or dwells with some function
states <- c(0, 1, 0, 1, 0, 1, 0, 1, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.0, 1.1, 0.6, 1.1, 0.8, 1.1)
my_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="example_segment")
segment.verify(my_burst)

# Now, a bad burst with two adjacent open dwells
states <- c(0, 1, 0, 1, 1, 0, 1, 0, 1)
dwells <- c(0.1, 1.1, 0.5, 0.2, 1.1, 0.6, 1.1, 0.8, 1.1)

# This will issue a warning
faulty_burst <- segment.create(states, dwells, seg=1, start_time=3.14159, name="faulty_segment")

# This will differentiate good and faulty bursts
segment.verify(faulty_burst)

# If you have a list of bursts, you can select the good ones with
# vbursts <- bursts.select(bursts, segment.verify)

util.basename

Remove suffix and path from filename.

Description

Remove suffix and path from filename.

Usage

util.basename(filename)

Arguments

filename string to extract basename from
Value

Name with suffix and path removed

Examples

util.basename("bursts/60uM-2017-08-18-16-32/60uM-712.dwt")
Index

bursts.check_subconductance, 3
bursts.conductance_states, 3
bursts.copy, 4
bursts.defined_by.tcrit, 4
bursts.get.gaps, 5
bursts.impose_deadtime, 6
bursts.modify_conductance, 7
bursts.pcloseds, 8
bursts.popens, 8
bursts.recombine, 9
bursts.remove.first.and.last, 10
bursts.select, 11
bursts.sort, 12
bursts.space.out, 13
bursts.start.times.update, 14
bursts.subconductance.as, 14

clampfit.read, 15
cplot.conductance_hist, 16
cplot.log_root_axes, 16
cplot.pclossed.ts, 17
cplot.popen.ts, 18
dwt.read, 18
dwt.write, 19
evt.extract_header, 20
evt.from.dwells, 20
evt.read, 21
evt.to.dwells, 22
evt.write, 22

hst.extract.header, 23
hst.read, 24
hst.write, 24

risetime.correct_gaussian, 25

scan.read, 26
segment.check_subconductance, 27
segment.closed_dwells, 27

segment.conductance_states, 28
segment.consecutives_to_dwells, 29
segment.copy, 29
segment.count.closed, 30
segment.count.dwells, 30
segment.count.open, 31
segment.create, 32
segment.duration, 33
segment.dwells_by_conductance, 33
segment.dwells_by_conductance_range, 34
segment.impose_deadtime, 35
segment.modify_conductance, 35
segment.name, 36
segment.open_dwells, 37
segment.pclosed, 38
segment.pconducance, 38
segment.pconducance_range, 39
segment.popen, 40
segment.seg, 40
segment.start_time, 41
segment.subconductance.as, 42
segment.verify, 42

util.basename, 43