Single Channel Burst Analysis with *scbursts*

Drummond et al.

2019-07-05

Contents

This guide is addressed to developers/programmers who might have to work with or extend scbursts. As programmers/developers can hopefully appreciate, this is a working document that will evolve as `scbursts` evolves. If any bugs are discovered, they can be reported on the github page, where they will hopefully be dealt with promptly.

This package is designed to extract information on the stochastic properties of single molecules. It was originally designed for dwell time analysis of single ion channel data derived from patch clamp experiments. It contains functions for importing and exporting idealized stochastic events; displaying, analyzing and correcting dwell durations; defining and sorting bursts, or clusters of bursts. `scbursts` can read and write, to and from, a variety of analysis software packages, including:

Program	Extension	URL
TAC	`evt`	bruxton.com/TAC/index.html
QUB	`dwt`	qub.mandelics.com/

Program	Extension	URL
SCAN	`txt`	github.com/DCPROGS
Clampfit	`xls`	moleculardevices.com/products/axon-patch-clamp-system/acquisition-and-analysis-software/pclamp-software-suite

`scbursts` is an open-source R package designed to be extensible – support for new file formats is easily added, and since the project is written in `R`, new functionality will be introduced in the future. Those interested in requesting or adding new functionality are invited to create an issue or pull request on the github page, or contact the dacosta]:[lab.

This manual gives a detailed overview on using `scbursts`. To learn more about the package, or if you would like to cite `scbursts`, please refer to the original publication. For details of specific functions, use `help()` or consult the technical manual (the source code is also available)

Contents

`.evts`, `.dwts`, `.txts`, and `.xls(x)`

`scbursts` can import data generated by a variety of single channel analysis suites, and thus in a number of different file formats, into the R environment. Data in these file formats is typically not native to the R environment, and so the relevant information (i.e. the sequence of dwell durations) must be extracted from each file as outlined below.

Handling `.evts`

In the case of TAC `evt` files, when importing files into the R environment, data transition through three states:

`evt file` → `Table of transition times` → `Table of dwells`

In order to import a TAC `evt` file, type:

```
# Load the library
library(scbursts)

infile <- "data/example1_tac.evt"

# Import the evt as a table
transitions <- evt.read(infile)

# Turn the transition times into dwells
dwells <- evt.to_dwells(transitions)

# With dwells defined, we can start doing an actual analysis
```

Writing to `evts` **is not symmetrical** with reading them, as the dwell times are automatically converted to transition times.

`Table of dwells` → `evt file`

To write an `evt`:

```r
# Write the corrected transition times to disk.
evt.write(dwells_corrected, file=file.path(tempdir(), "100uMc.evt"))
```

Handling QUB `.dwts`

`dwts` are lists of dwell times, and so they convert to **segments** and **bursts** very naturally. To read and write

```r
dwells <- dwt.read("example1_qub.dwt")

# ...
#
# Correct the dwells or do an analysis
#
# ...

dwt.write(corrected_dwells, file=file.path(tempdir(), "example1_qub_corrected.dwt"))
```

Handling SCAN files

Importing SCAN files is equally simple. The one caveat is that SCAN often produces binary files, which **`scbursts` cannot read**, however there is another DCPROGS tool to convert these to `txt` files.

```r
infile <- "data/example1_scan.txt"
record <- scan.read(infile)
head(record)
```

Handling Clampfit files

Importing clampfit files is also simple.

```r
infile <- "data/example1_clampfit.xlsx"
dwells <- clampfit.read(infile)
head(dwells)
```

Segments

`.dwt` files record sequences of open and closed dwell durations ("dwells"), where the single channel alternates between open (1) and closed (0) states (at least, they should. Though sometimes there are errors which cause recording to show more than a single 1 or 0 in succession. This will be discussed later). There can be multiple "segments" of dwells, each corresponding to a continuous stretch of idealized events.

```
Segment: 1   Dwells: 4181
    1   0.000150
    0   0.000900
    1   0.078490
    0   1.910400
    1   0.421490
.
.
.
    0   1.334670
    1   0.012270
Segment: 2   Dwells: 7653
    1   0.065900
    0   0.596160
    1   0.849920
    0   0.023830
    1   0.612380
    0   0.022120
.
.
.
```

segment is also the name of the data-type `scbursts` defines (refer to `segment.R`), so that reading a `dwt` gives you a list of `segments` (which is an `R` object, like a `data.frame`). Often one has that each segment is meant to correspond to a burst, and so when we say “bursts”, typically we refer to a list of segments. As a consequence, there are also functions in `scbursts` such as `bursts.select` which make working with these lists more convenient.

The reason `dwts` yield a list of segments is because there are often several continuous stretches of recording separated by pauses.

```r
# Load the library
library(scbursts)

# Import a pre-packaged file (stored inside the folder extdata)
dwt_example <- system.file("extdata", "example1_qub.dwt", package = "scbursts")

# Import the evt as a table
dwells <- dwt.read(dwt_example)
length(dwells)
```

```
## [1] 1
```

```r
# Transition times and states of first segment (units are in seconds)
head(dwells[[1]])
```

```
##   states   dwells
## 1      1 0.001380
## 2      0 0.000034
## 3      1 0.001759
## 4      0 0.000312
## 5      1 0.001201
## 6      0 0.000535
```

There are a number of functions that act on segments. The functions pertaining to segments start with `segment.`, *some* of the available functions are:

Function	Description
`segment.open_dwells`	Extract open dwells as a vector
`segment.closed_dwells`	Extract closed dwells as a vector
`segment.count_open`	Count number of openings
`segment.count_closed`	Count number of closed
`segment.popen`	Empirical P(Open) for segment
`segment.pclosed`	Empirical P(Closed) for segment
`segment.duration`	Total segment duration
`segment.verify`	Is the segment error free?

An important point: segments are not just vectors, they also store meta-data that includes when the segment took place in the recording. This allows for replotting of the segments later in a way that is consistent with what the original time-series would have looked like. An

example of this will be seen in the plots at the end of this document.

Bursts

In the Single Channel community, continuous stretches of open and closed dwells that correspond to the activity of a single ion channel are often referred to as **bursts** of single channel activity. **Bursts** are usually defined by a critical closed duration (t_{crit}), which is determined from analysis of a closed dwell duration histogram. This t_{crit} stipulates that openings separated by closings briefer than t_{crit} originate from the same burst of single-channel activity, while openings separated by closings longer than t_{crit} originate from different bursts (see Chapter 19, Section 5.5.1 of Colquhoun and Sigworth (1995)).

An idiosyncrasy of the package has to be mentioned: In scbursts, the scientific notion of **bursts** and the program definition of `bursts` *usually* coincide, but not always. As mentioned, a `segment` denotes a contiguous sequence of idealized dwells, however idealized single channel recordings can be made up of one **or more** segments. Furthermore, each segment can contain multiple **bursts** (in the scientific sense). So the unfortunate side effect of this is that `scbursts` imports data files like `evts` and `dwts` as a list of recordings, which is a list of segments, so they "look like" `bursts`. Below is a little example of this: the problem is easily resolved; one just has to pass this list through a burst detector. So, for example, a function like `bursts.defined_by_tcrit` will break up the recordings into lists of bursts (using an input critical time).

```r
# Load the library
library(scbursts)

# Import a pre-packaged file (stored inside the folder extdata)
infile <- system.file("extdata", "example_multiple_segments.dwt", package = "scbursts")

# Import the evt as a table
records <- dwt.read(infile)
```

```
## Warning in dwt.read(infile): Burst (or record) 1 seems to have been
## misrecorded!
```

```r
# The number of records
length(records)
```

```
## [1] 2
```

```r
# Correct the risetime (= Tr) (default time in seconds)
records_c <- risetime.correct_gaussian(Tr=35.0052278,records, units="us")


# Define critical time (tcrit=100 ms)
# However, now we can use a `bursts` function, `bursts.defined_by_tcrit`
# to turn the records into actual bursts
bursts <- bursts.defined_by_tcrit(records_c , 100, units="ms")
```

```
## Warning in bursts.defined_by_tcrit(records_c, 100, units = "ms"): Merging
## all recordings into one recording. A large (but arbitrary) amount of time
## will seperate the recordings.
```

```
## Warning in bursts.defined_by_tcrit(records_c, 100, units = "ms"): Burst 24 seems to have been misrec
```

```r
# The number of bursts
length(bursts)
```

```
## [1] 36
```

```r
## Now you can carry out analysis of the bursts
```

Some comments on the warning messages:

1. The first warning message informs you that there is a recording error present in the first part of the recording. How this happens and what it means is explained later in this document.
2. The second warning message informs you that two or more records were in the file, and they are getting concatenated together. So burst `n` might belong to the first record, and burst `n+1` might belong to the second record.
3. The third warning is a return of the first warning. The recording error from the first record was inside of burst 24. In light of this, we can just delete that burst and then proceed with our analysis

Since **bursts** are attributable to the activity of a single channel, **bursts** are the units that we are interested in for kinetic analysis of single ion channels. `scbursts` can perform many functions on **bursts**.

Taking a subset

We might not always be interested in all the bursts, but perhaps bursts that have some characteristic, such as a high *P(Open)*. For instance, often one wants to discard aberrant (our outlier) bursts prior to kinetic analysis. So, rather than using the full list of bursts, we might want to extract a few bursts. Here's how you do that:

```r
# Load the library
library(scbursts)

# Import a pre-packaged file (stored inside the folder extdata)
infile <- system.file("extdata", "example1_tac.evt", package = "scbursts")

# Import the evt as a table
tables <- evt.read(infile)
records <- evt.to_dwells(tables)

# Correct the risetime (default time in seconds)
records_c <- risetime.correct_gaussian(Tr=35.0052278,records, units="us")

# Define critical time (tcrit=100 ms)
bursts <- bursts.defined_by_tcrit(records_c , 100, units="ms")

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

high_bursts <- bursts.select(bursts, high_popen)
```

This will return a list of bursts which you can work on, and you can remove more bursts as you wish until you have what you're looking for. If you want to extract bursts and write them to file (for instance, for processing by MIL/QUB), you can use the line

```r
high_bursts <- bursts.select(bursts, high_popen, one_file=TRUE)
```

and this will extract the high P(Open) bursts **as a single segment**, which can then be written to a single `dwt` file. The advantage of using this trick, is that this particular function can write the bursts to a single segment **and preserve the amount of time that actually took place between the bursts**. (In order to do this manually, you have to use `bursts.recombine`.)

An Example: Correcting Recording Errors

As a specific example, when using actual recorded data, there are sometimes errors where multiple openings or closings are recorded in succession

```
Segment: 1   Dwells: 4181
    1   0.000150
    0   0.000900
    1   0.078490
    1   0.046750    <- this doesn't make sense
    1   0.037790
    0   1.910400
    1   0.421490
    0   1.896120
.
.
.
```

this is obviously *physically* impossible, but sometimes appears in the data. To correct this, one would separate the recording into bursts (as usual) and then remove the burst where the error occurred.

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

# This will raise a warning message to alert you that the data has problems
dwells <- dwt.read(infile)
```

```
## Warning in dwt.read(infile): Burst (or record) 1 seems to have been
## misrecorded!
```

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

# This will also raise a warning message to alert you that specific bursts have problems
bad_bursts <- bursts.defined_by_tcrit(dwells_c, 100, units="ms")
```

```
## Warning in bursts.defined_by_tcrit(dwells_c, 100, units = "ms"): Merging
## all recordings into one recording. A large (but arbitrary) amount of time
## will seperate the recordings.

## Warning in bursts.defined_by_tcrit(dwells_c, 100, units = "ms"): Burst 24 seems to have been misreco
```

```
length(bad_bursts)
```

```
## [1] 36
```

```
# This will remove the problems. It will leave only the good bursts.
fixed_bursts <- bursts.select(bad_bursts, segment.verify)

length(fixed_bursts)
```

```
## [1] 35
```

In a related problem, sometimes one might want to discard the first and last burst, as you might now know whether or not you began recording in the middle of a burst (so the depiction of burst behavior would be inaccurate). You can fix this with `bursts.remove_first_and_last`.

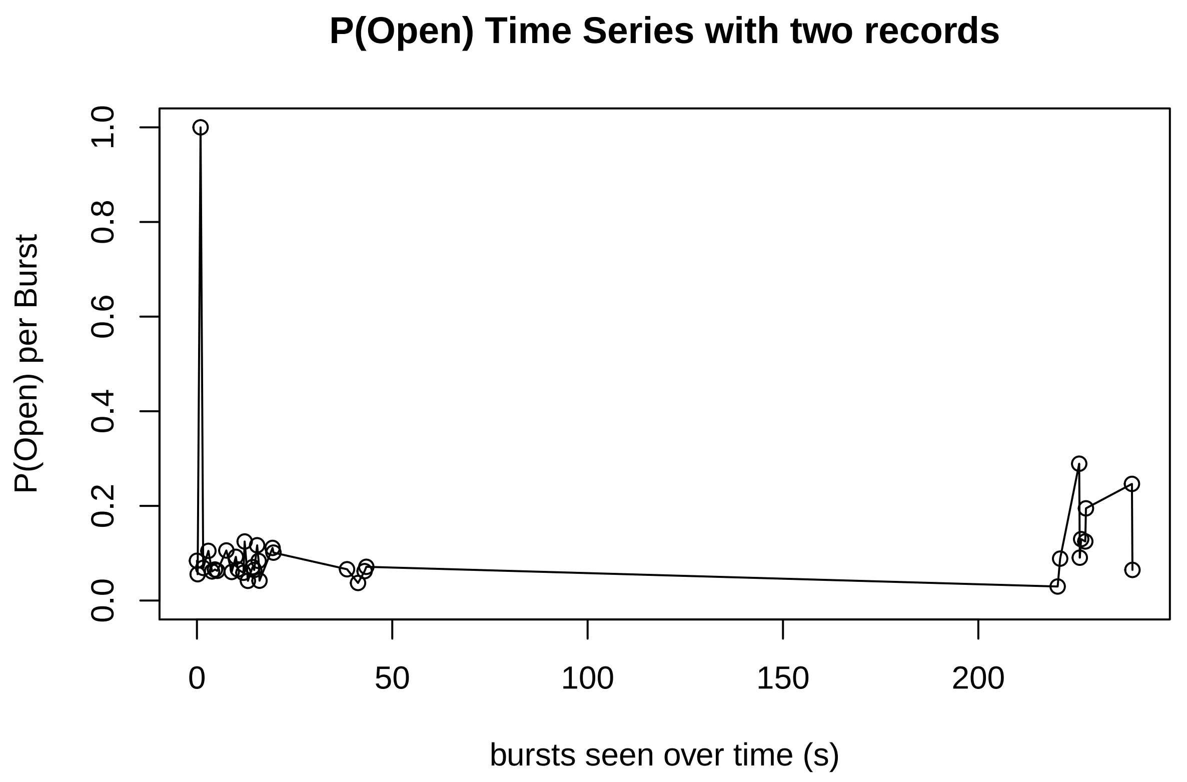
(Advanced) Writing bursts back to files

We mentioned that we could take a segment, split it into bursts, remove some bursts, and then write back to file. We mentioned a short-cut to do this, but also that this could be done manually. This requires the use of `bursts.recombine`. The tool isn't trivial, because it preserves the elapsed time between the bursts when rejoining them. But, we also mentioned that recordings could contain multiple segments, but what was not emphasized was that this means that **we usually have no idea how much time transpires between segments**. However, despite this, there are times where we want to merge all bursts as though they were occurring in one segment. For this reason, **we need a function to artificially insert gaps between bursts**. We often use these in conjunction, with something like

```r
# If you have multiple records, you can recombine them with
# This is now just one list of spaced out segments.
records <- bursts.space_out(records, sep_factor=1000)
record <- bursts.recombine(records)
```

The different recording segments will have a substantial amount of time between them, which should be discernible by eye. For example:

```r
# From example_multiple_segments.dwt
times  <- sapply(fixed_bursts, segment.start_time)
popens <- sapply(fixed_bursts, segment.popen)

plot(times,popens, main="P(Open) Time Series with two records", ylab="P(Open) per Burst",
xlab="bursts seen over time (s)", ylim=c(0,1))
lines(times, popens)
```



There are scenarios where this is probably the most natural way to create a single segment with all the data you're interested in. If you are interested in playing around with the spacing between bursts (for example, the closings that exceeded a critical time), you can also look at `bursts.get_gaps` which extracts this information as a vector.

NOTE: Because of the –arbitrary– long gaps between bursts, the time series data produced in this way cannot be used for some types of analyses. One should be aware of this and know when to conduct analysis on a single recording segment instead of over all recordings at once. The reader who wants to change the spacing between records may look at

`bursts.start_times_update`

Sorting and more

In addition to taking a subset of bursts according to some criteria, one might want to sort bursts according to some metric, or simply get tabulated values of some metric. For example, what is the average P(Open) across all bursts? Or what do the bursts look like when we rank them by P(Open)?

Sorting is implemented by `bursts.sort`. It requires only a metric on segments - a function which takes a segment and gives you a number. For example:

```
# Create a list of bursts, sorted by your chosen function
sorted <- bursts.sort(bursts, segment.popen, reverse=TRUE)

# In some cases, it might be that multiple bursts share the same value
# and so the "order" is a bit arbitrary in those cases.
sorted[[1]]
```

```
##   states       dwells
## 1      1 3.232048e-05
```

as for collecting data on all bursts, `bursts.popens` and `bursts.pcloseds` have been provided for convenience, and so you could get the average with

```
mean(bursts.popens(bursts))
```

```
## [1] 0.778477
```

But, it isn't hard to write your own functions that do this. The definition of `bursts.popens` is simply

```
bursts.popens <- function (bursts) { sapply(bursts, segment.popen) }
```

You can simply use `sapply` in an analogous way with any function that deals with a single segment. For instance, you could find the average duration with

```
mean(sapply(bursts, segment.duration))
```

```
## [1] 0.05363405
```

Working with bursts v.s. segments

Most important functions can be applied either to a single segment, or to a list of segments. For instance, the following two are equivalent:

```
# Correct the risetime

corrected_records <- list()
for (i in 1:length(records)) {
    corrected_records[[i]] <- risetime.correct_gaussian(Tr=35.0052278, records[[i]], units="us")
}

# Write the corrected record to a .dwt file
dwt.write(corrected_records, file=file.path(tempdir(), "example1_qub_corrected.dwt"))
```

and this simplified code

```
# Correct the risetime
records_c <- risetime.correct_gaussian(Tr=35.0052278, records, units="us")
```

```r
# Write the corrected record to a .dwt file
dwt.write(records_c, file=file.path(tempdir(), "example1_qub_corrected.dwt"))
```

but this is not always true, and not every function can be made to work on either. You may have to write your own functions, use for loops, or use `sapply` (or `lapply`) in order to do everything that you want to do.

Risetime Correction

It was skimmed over, but risetime correction on the recordings can be accomplished simply with

```r
# Load the library
library(scbursts)

# Import a pre-packaged file (stored inside the folder extdata)
infile <- system.file("extdata", "example1_tac.evt", package = "scbursts")


# Import the evt as a table
tables <- evt.read(infile)

# Turn the transition times into dwells
records <- evt.to_dwells(tables)

# Correct the risetime (default time in seconds)
records_c <- risetime.correct_gaussian(Tr=35.0052278,records, units="us")
```

```r
evt.write(records_c, file=file.path(tempdir(), "example_corrected.evt"))
```

As the name of the function suggests, the current risetime correction attempts to undo the effects of a Gaussian filter. This method might not be optimal, and may be replaced later. For a more detailed explanation, see Section 4.1.1 of Colquhoun and Sigworth (1995).

Plotting

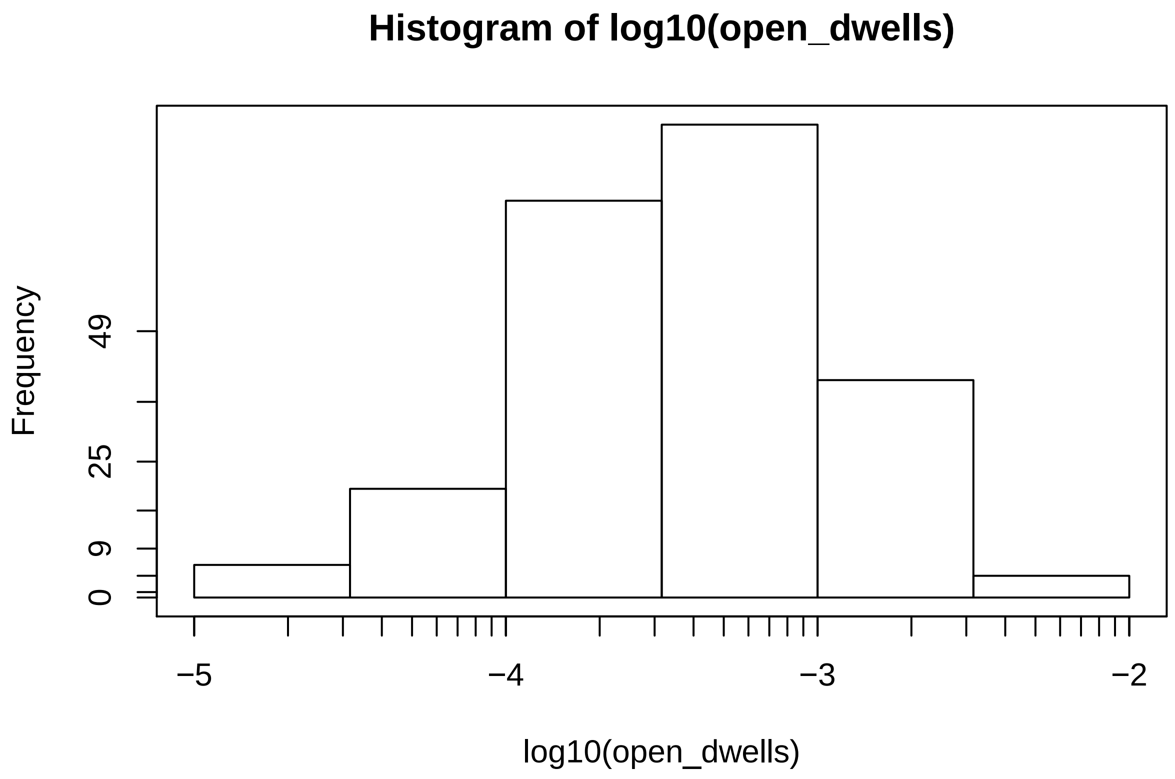
Here are example of a few common plots one might want.

```r
library(scbursts)

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

# Skipping risetime correction
bursts <- bursts.defined_by_tcrit(dwells,92,units='ms')

# We will be plotting with this
record <- bursts.recombine(bursts)
```

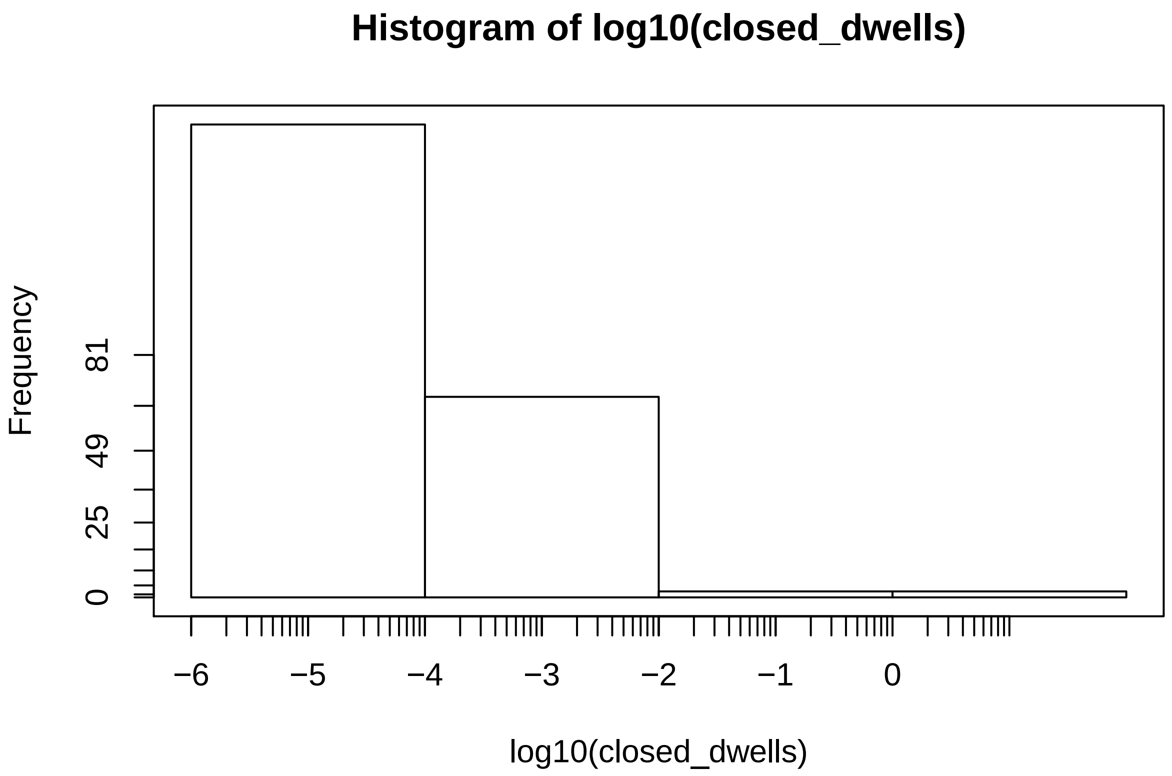
NOTE: If you merge multiple records into one, you might artificially add some –huge– closed dwells separating bursts. These will add a few high closed-times the histogram.

Open times and closed times (uncorrected)

```
open_dwells <- segment.open_dwells(record)
hist(log10(open_dwells), axes=FALSE, breaks=length(record$dwells)/100)
cplot.log_root_axes(open_dwells)
```



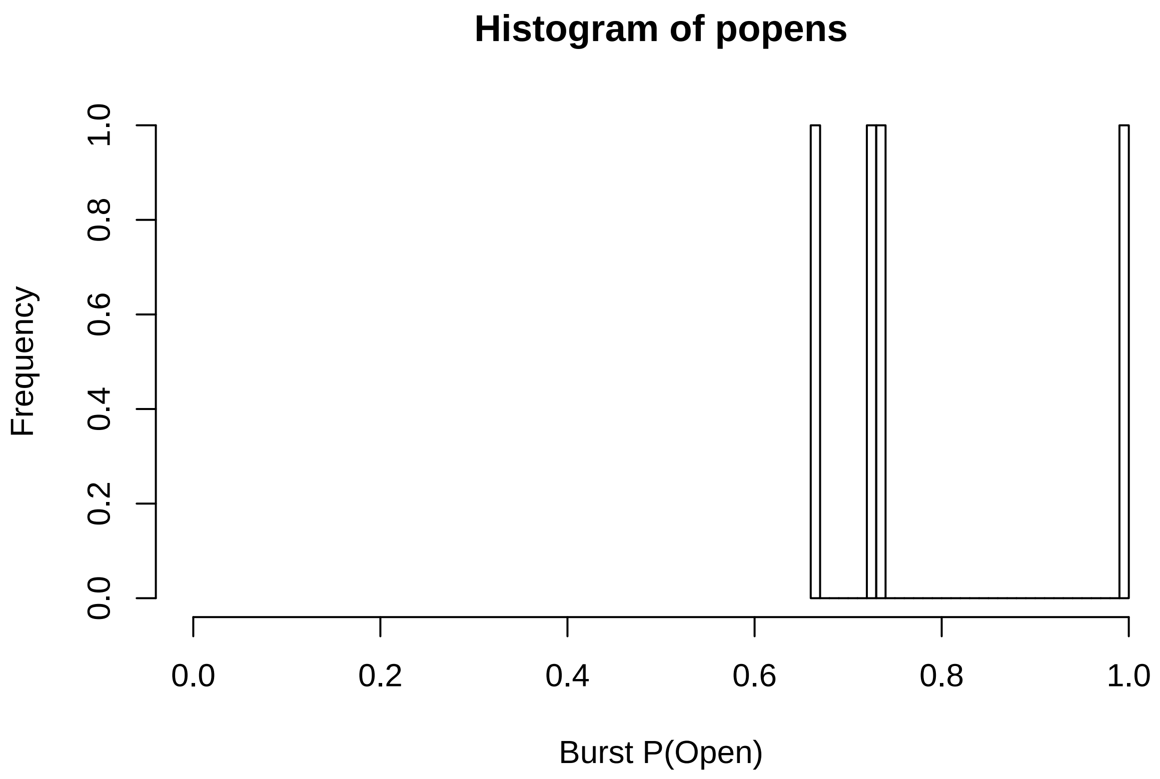
```r
closed_dwells <- segment.closed_dwells(record)
hist(log10(closed_dwells), axes=FALSE, breaks=length(record$dwells)/100)
cplot.log_root_axes(closed_dwells)
```



The outlier on the far right may be due to merging multiple records into one, as mentioned earlier.

P(Open) (uncorrected)

```r
popens <- bursts.popens(bursts)
hist(popens, xlab="Burst P(Open)", breaks=30, xlim=c(0,1))
```

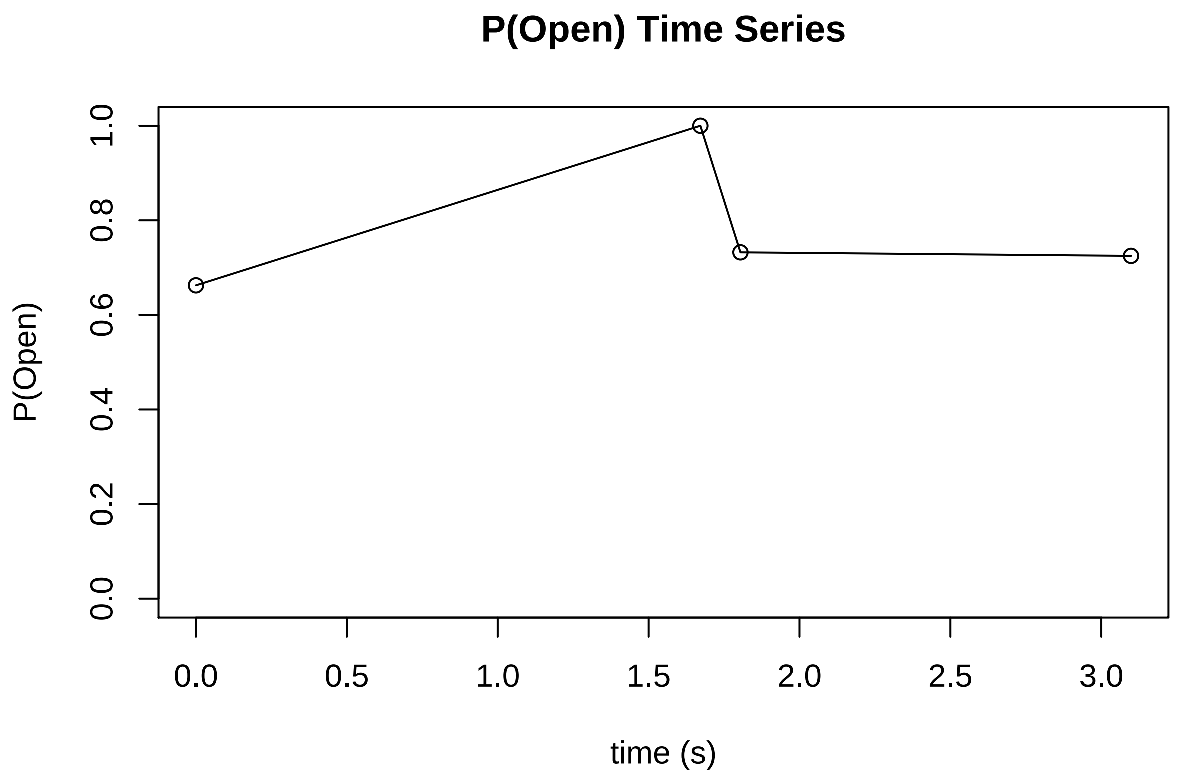


P(Closed) is similar.

Time Series (uncorrected)

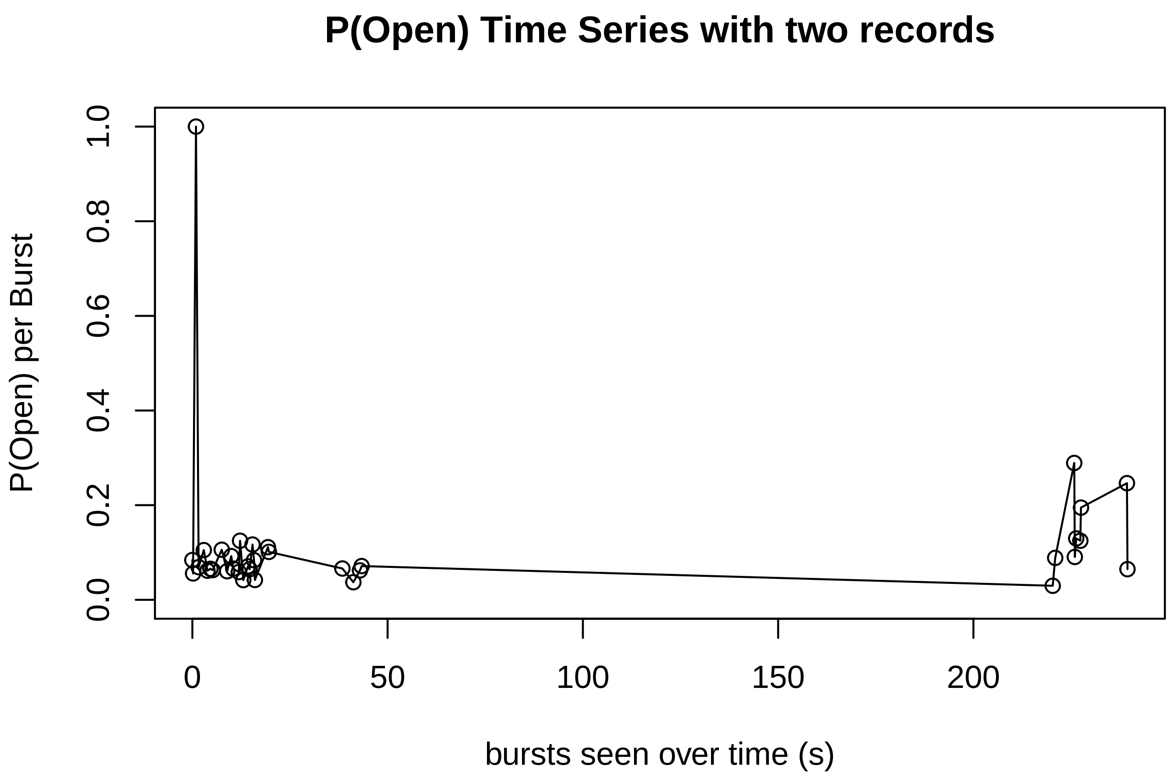
A basic example

```
# To make this more visible, you can also export it as a large `.png` file
cplot.popen_ts(bursts)
```



`cplot.popen_ts` is just for convenience, you can also just just manually do the plotting. Also, `scbursts` automatically handles data that contains multiple records by inserting a comically large amount of time between records. `scbursts` issues warnings when this happens, and one should heed those warnings. The time-series data produced in this way is suitable for visualization, but **it is not suitable for fitting some types of models**. Just be wary of this.

```
# From example_multiple_segments.dwt
times  <- sapply(fixed_bursts, segment.start_time)
popens <- sapply(fixed_bursts, segment.popen)

plot(times,popens, main="P(Open) Time Series with two records", ylab="P(Open) per Burst",
xlab="bursts seen over time (s)", ylim=c(0,1))
lines(times, popens)
```



Subconductive States

`scbursts` also has support for data with subconductive states (though on reading in a file, `scbursts` will issue a warning to inform the user that their data has subconductive states). There are tools available for visualizing the different subconductive states (`cplot.conductance_hist`) and functions for manipulating subconductive states (for instance, merging two or more states together, or removing subconductive states entirely). Some functions for this are:

Function	Description
`segment.conductance_states`	Get list of conductance states
`segment.check_subconductance`	Check for subconductance
`segment.subconductance_as`	Replace subconductive states with 0 or 1
`segment.modify_conductance`	Remove or change conductance states
`bursts.conductance_states`	Get list of conductance states
`bursts.check_subconductance`	Check for subconductance

Function	Description
`bursts.subconductance_as`	Replace subconductive states with 0 or 1
`bursts.modify_conductance`	Remove or change conductance states
`cplot.conductance_hist`	Plot the conductance states

An example is also below.

Example Workflows

DWT

```r
library(scbursts)

infile <- system.file("extdata", "example1_qub.dwt", package = "scbursts")
dwells <- dwt.read(infile)
# dwells <- dwt.read('example1.dwt')

bursts <- bursts.defined_by_tcrit(dwells,100,units='ms')
twoplus <- function(seg){
        return(segment.count_open(seg)>=2)
}
bursts_twoplus <- bursts.select(bursts,twoplus)
head(dwells[[1]])
```

```
##   states   dwells
## 1      1 0.001380
## 2      0 0.000034
## 3      1 0.001759
## 4      0 0.000312
## 5      1 0.001201
## 6      0 0.000535
```

EVT

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

# transitions <- evt.read("July28-6.evt")

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

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

head(dwells_c[[1]])
```

```
##   states       dwells
## 1      1 1.360730e-03
## 2      0 5.352284e-05
## 3      1 5.173524e-05
## 4      0 2.080985e-05
## 5      1 2.180100e-04
## 6      0 2.105737e-05
```

SCAN

```
library(scbursts)

infile <- system.file("extdata", "example1_scan.txt", package = "scbursts")
dwells <- scan.read(infile)
dwells_c <- risetime.correct_gaussian(Tr=35.0052278, dwells, units="us")

bursts <- bursts.defined_by_tcrit(dwells_c,100,units='ms')
twoplus <- function(seg){
        return(segment.count_open(seg)>=2)
}
bursts_twoplus <- bursts.select(bursts,twoplus)
head(bursts_twoplus[[1]])
```

```
##   states       dwells
## 1      1 1.352536e-03
## 2      0 6.247068e-05
## 3      1 4.103552e-05
## 4      0 2.743168e-05
## 5      1 4.308123e-04
## 6      0 2.606283e-05
```

QUB

```
library(scbursts)

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')
twoplus <- function(seg){
        return(segment.count_open(seg)>=2)
}
bursts_twoplus <- bursts.select(bursts,twoplus)
head(bursts_twoplus[[1]])
```

```
##   states      dwells
## 1      1 1.38000e-03
## 2      0 3.45043e-05
## 3      1 1.75900e-03
## 4      0 3.12000e-04
## 5      1 1.20100e-03
## 6      0 5.35000e-04
```

Clampfit

```
library(scbursts)
```

```
infile <- system.file("extdata", "example1_clampfit.xlsx", package = "scbursts")
dwells <- clampfit.read(infile)
```

```
## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
## * `` -> ...5
## * ... and 6 more problems
```

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

bursts <- bursts.defined_by_tcrit(dwells_c,100,units='ms')
twoplus <- function(seg) {
    return(segment.count_open(seg)>=2)
}
bursts_twoplus <- bursts.select(bursts,twoplus)
head(bursts_twoplus[[1]])
```

```
##   states       dwells
## 1      1 1.359000e-03
## 2      0 5.400240e-05
## 3      1 5.000919e-05
## 4      0 2.167236e-05
## 5      1 2.160000e-04
## 6      0 2.207933e-05
```

Example With Subconductive States

```
library(scbursts)

infile <- system.file("extdata", "example4.dwt", package = "scbursts")
dwells <- dwt.read(infile)
```

```
## Warning in dwt.read(infile): Burst (or record) 1 has subconductive states!
```

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

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

```
## Warning in bursts.defined_by_tcrit(dwells_c, 100, units = "ms"): Burst (or record) 1 has subconducti
```

```
bursts.conductance_states(bursts)
```

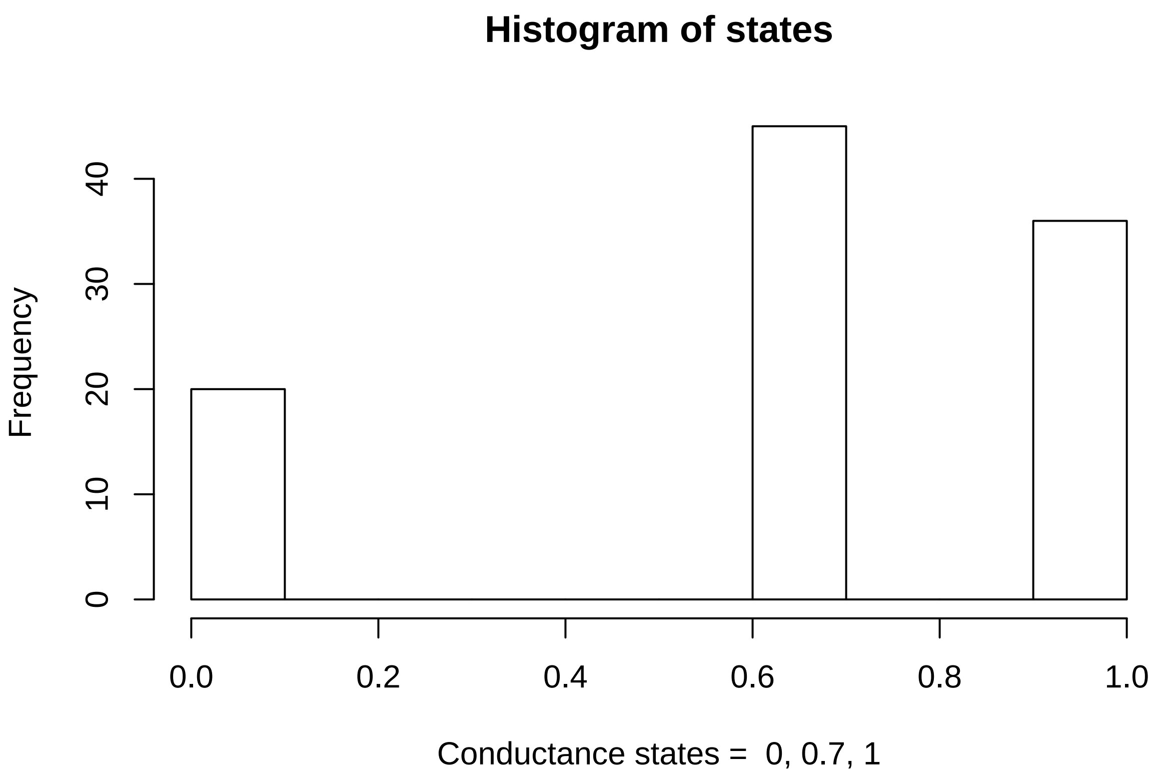
```
## [1] 0.0 0.7 1.0
```

```
cplot.conductance_hist(bursts)

head(bursts[[1]])
```

```
##   states      dwells
## 1    0.7 1.39298e-03
## 2    1.0 2.99420e-04
## 3    0.7 1.94308e-03
## 4    0.0 1.88507e-05
## 5    0.7 1.12071e-03
## 6    1.0 1.16738e-03
```

```
bursts_d <- bursts.subconductance_as(bursts, "open")

head(bursts_d[[1]])
```

```
##   states       dwells
## 1      1 3.635480e-03
## 2      0 1.885070e-05
## 3      1 2.288090e-03
## 4      0 1.623060e-03
## 5      1 4.169384e-02
## 6      0 2.490355e-05
```



References

Colquhoun, David, and FJ Sigworth. 1995. "Fitting and Statistical Analysis of Single-Channel Records." In *Single-Channel Recording*, 483–587. Springer.