

Package ‘rethomics’

June 8, 2015

Type Package

Title Rethomics, a package to analyse high-throuput animal behaviour data.

Version 0.1

Date 2015-01-29

Author Quentin Geissmann

Maintainer Quentin Geissmann <qgeissmann@gmail.com>

Description This package was primarily developed to study sleep and circadian rhythm in fruit flies in combination with pysolovideo python package.

License GPL (>=3.0)

Depends R (>= 2.15.0),
MASS,
RSQLite,
data.table,
pracma,
zoo,
ggplot2

LazyLoad TRUE

Collate 'io.R'
'interpol.R'
'plots.R'
'sleep.R'
'utils.R'

URL <https://github.com/gilestrolab/rethomics>

R topics documented:

| | |
|---------------------------------|----|
| boutAnalysis | 2 |
| days | 2 |
| ethogramPlot | 3 |
| fetchPsvResultFiles | 4 |
| hours | 5 |
| loadDAMFile | 5 |
| loadMetaData | 6 |
| loadPsvData | 7 |
| loadSampleData | 9 |
| maxVelocityClassifier | 10 |

2

daysmins 10overviewPlot 11sleepAnnotation 12virtualBeamCrossClassif 13Index 14

| | |
|--------------|-------------|
| boutAnalysis | <i>Todo</i> |
|--------------|-------------|

Description

Todo

Usage

boutAnalysis(var, data)

| | |
|------|---|
| days | <i>Trivially converts days to seconds</i> |
|------|---|

Description

Trivially converts days to seconds

Usage

days(x)

Arguments

| | |
|---|-------------------|
| x | number of seconds |
|---|-------------------|

Value

the corresponding number of seconds

See Also

[hours mins](#)

| | |
|--------------|--|
| ethogramPlot | <i>Displays the temporal and inter-individual average of a variable of interest.</i> |
|--------------|--|

Description

This function produces a graph where the variable of interest and time are on the y and x axes, respectively. It can be used to visualise temporal trends per groups of conditions. The response variable, y, is grouped by time windows of defined size.

Usage

```
ethogramPlot(y, data, condition = NULL, summary_time_window = mins(30),
             normalise_var_per_id = FALSE, error_bar = NULL)
```

Arguments

| | |
|----------------------|--|
| y | The variable of interest. |
| data | The data.table containing the data. It must have a column with the same name as y. |
| condition | An optionnal grouping factor to order rows. |
| summary_time_window | the width (in seconds) of the time window used to draw each “pixel”. |
| normalise_var_per_id | whether each row is to be normalised (using $\text{new_x} = (x - \text{mean}(x))/\text{sd}(x)$). |
| error_bar | what type of error bar should be used. It should be one of NULL, ‘sem’ or ‘sd’. |

Value

A ggplot object that can be plotted directly, or modified.

See Also

[overviewPlot](#) to show per-individual patterns

Examples

```
# Load sample data
data(monitor_validation)
# We perform sleep annotation on all animals
my_data <- monitor_validation[,sleepAnnotation(.SD),by=key(monitor_validation)]
# No condition
p <- ethogramPlot(asleep,my_data)
print(p)
# We make a dummy male vs female condition;
# males could be, for instance, in any even region
my_data[,sex:=ifelse(region_id %% 2==0,"male","female")]
p <- ethogramPlot(asleep,my_data,condition = sex)
print(p)
p <- ethogramPlot(asleep,my_data,condition = sex,error_bar="sem")
print(p)
# p is simply a ggplot object, so we can change things:
print(p + labs(title="MY own title"))
```

| | |
|---------------------|--|
| fetchPsvResultFiles | <i>Query files from a PSV data directory according to the date of the experiment and the device which acquired the data.</i> |
|---------------------|--|

Description

This function is designed to list and select experiemental files. In general, end-users will want to retrieve path to their experimental files according to the date and ID of the video monitor without having to understand the underlying directory structure.

Usage

```
fetchPsvResultFiles(result_dir, query = NULL)
```

Arguments

| | |
|------------|---|
| result_dir | The location of the result directory (i.e. the folder containing all the data). |
| query | An optionnal query formatted as a dataframe (see details). |

Details

The optionnal argument query is expected to be a table where every row maps an experiment. In many respects, it is similar to the what argument in [loadPsvData](#). The only difference is that it does not have a path column. Instead, it must contain two columns:

- **time** The date and time when the experiment started formatted either as 'yyyy-mm-dd' or 'yyyy-mm-dd_hh:mm:ss'. In the former case, there may be several matching experiments to a single time (starting the same day). When this happens, *only the last* is returned, and a warning message is displayed.
- **machine_name** The name of the machine that acquired the data.

The result is meant to be used directly, as the what argument, by [loadPsvData](#) (see examples).

Value

The query extended with the requested paths. When query is not specified, the function returns a table with all available files.

Note

PSV stores data in a hard-coded directory structure `/root_dir/machine_id/machine_name/datetime/file.db`, where:

- **machine_id** Is, in principle, a universally unique identifier of the acquisition device.
- **machine_name**, a human friendly name for acquisition device. In practice, this is expected to be unique within laboratory.
- **datetime**, the date and time of the start of the experiment

Examples

```
## Not run:
# This is where I store the data on my computer
MY_DATA_DIR <- "/data/psv_results/"

query <- data.table(date="2015-06-02",
                    machine_name=c("GGSM-001", "GGSM-003"),
                    region_id = rep(1:10, each=2))

print(query)
map <- fetchPsvResultFiles(MY_DATA_DIR, query)
dt <- loadPsvData(map)

## End(Not run)
```

| | |
|-------|--|
| hours | <i>Trivially converts hours to seconds</i> |
|-------|--|

Description

Trivially converts hours to seconds

Usage

```
hours(x)
```

Arguments

x number of seconds

Value

the corresponding number of seconds

See Also

[days mins](#)

| | |
|-------------|---|
| loadDAMFile | <i>Read a text file formatted as DAM2 into a single data table.</i> |
|-------------|---|

Description

This function is used to load data from DAM2 devices as a data.table.

Usage

```
loadDAMFile(FILE, channels = NULL, min_time = 0, max_time = Inf,
            interval = 60)
```

Arguments

| | |
|----------|--|
| FILE | the name of the input file. |
| min_time | exclude data before min_time (in seconds). This time is relative to the start of the experiment. |
| max_time | exclude data after max_time (in seconds). This time is relative to the start of the experiment. |

Value

If rois has only one element, a dataframe. Otherwise, a list of dataframes (one per ROI)

Note

Analysis of many long (several days) recording can use a large amount of RAM. Therefore, it can sometimes be advantageous to load an process ROIs one by one.

See Also

[loadMetaData](#) To display global informations about the experiment.

Examples

```
## Not run:
FILE <- "Monitor53.txt"
out <- loadDAMFile(FILE)
#histogram of x marginal distribution
hist(out[roi_id == 1, x], nclass=100)

## End(Not run)
## Not run:
# More realistic example where we have experimental conditions, and
# we want to resample data at 1.0Hz.
# First, the conditions:
conditions <- cbind(roi_id=1:32, expand.grid(treatment=c(T,F), genotype=LETTERS[1:4]))
print(conditions)

## End(Not run)
```

loadMetaData

Retrieve metadata from a result file.

Description

This function is used to obtain metadata – such as ‘time and date of the experiment’, ‘acquisition device’, ‘version of the software’ and such – embedded in a result file generated by PSV.

Usage

```
loadMetaData(FILE)
```

Arguments

| | |
|------|-----------------------------|
| FILE | the name of the input file. |
|------|-----------------------------|

Value

A list containing fields for metadata entries

See Also

[loadPsvData](#) to load raw data.

Examples

```
## Not run:
FILE <- "result.db"
out <- loadMetaData(FILE)
names(out)

## End(Not run)
```

| | |
|-------------|--------------------------------------|
| loadPsvData | <i>Read data from a result file.</i> |
|-------------|--------------------------------------|

Description

This function is used to convert all the information contained in a result file generated by PSV (i.e a .db file) into an R ‘data.table’.

Usage

```
loadPsvData(what, min_time = 0, max_time = Inf, reference_hour = NULL,
  verbose = TRUE, FUN = NULL, ...)
```

Arguments

| | |
|----------------|---|
| what | an object describing which file(s) to load and, optionally, associated variables/conditions (see details). |
| min_time | exclude data before min_time (in seconds). This time is relative to the start of the experiment. |
| max_time | exclude data after max_time (in seconds). It is also relative to the start of the experiment. |
| reference_hour | the hour, in the day, to use as t_0 reference. When unspecified, time will be relative to the start of the experiment. |
| verbose | whether to print progress (a logical). |
| FUN | an optionnal function to transform the data from each ‘region’ (i.e. a data.table) immediatly after it has been loaded. |
| ... | extra arguments to be passed to FUN |

Details

what can be one of two objects:

- A character vector. In which case, it is assumed that each element is the path to a different file to load.
- A dataframe. The dataframe *must* have a column named 'path'. The path basename will be used as a unique identifier for a specific experiment (experiment_id). Arbitrary column can be added to map experimental conditions to file name. In addition, the dataframe can have a column named region_id. When defined, only the specified combinations of path and region_id will be loaded. This allows to map additional conditions (i.e. data frame columns) to specific regions/files. When additional conditions are provided, they will result in creation of custom columns in the output of this function.

Value

A data.table where every row is an individual measurement. That is a position at a unique time (t) in a unique region (region_id), and from a unique result file/experiment (experiment_id). The time is expressed in seconds. Distance units (e.g. xy position, height/width) are expressed as a fraction of the width of the region they originate from.

See Also

[loadMetaData](#) To display global informations about a specific file.

Examples

```
# First of all, let us load files from the data sample included within this package.
# Most likely, you will already have your own data files.
sample_files <- c("tube_monitor_validation_subset.db",
                  "monitor_validation_subset.db")
# Extract the files in your computer
paths <- sapply(sample_files, loadSampleData)
# Now, 'paths' is just a vector of file names:
print(paths)
#####
#####
# Case 1: load ALL REGIONS from a SINGLE FILE
validation_data_file <- paths[1]
# 'validation_data_file' is simply the path to the .db file in your computer
dt <- loadPsvData(validation_data_file)
print(dt)
#####
# Case 2: load ALL REGIONS from MULTIPLE FILES
# we pass all the files we want to load as the 'what' argument
dt <- loadPsvData(paths)
# Note the column 'experiment_id' in dt. It tells us which file/experiment
# each measurement originates from.
print(dt)

#####
# Case 3: load ALL REGIONS from MULTIPLE FILES AND add CONDITIONS
# Let us imagine that each file/experiment
# was acquired under different experimental condition.
# We can encode this information in a 'master-table' (i.e a data.frame)
# in which a column named \code{path} maps experimental condition(s).
```



```

# For instance, 2 different treatments:
master_table <- data.frame(path=paths, treatment=c("control", "drug_A"))
# Let us check our table:
print(master_table)
# The table looks OK, so we load the actual data
dt <- loadPsvData(master_table)
# Note that `dt` now contains a column for your treatment.
print(colnames(dt))
# This makes it easier to perform things such as average per treatment.
print(dt[,.(mean_x = mean(x)),by="treatment"])
#####
# Case 4: load SELECTED REGIONS from MULTIPLE FILE, WITH CONDITIONS
# Sometimes, different regions contain different conditions.
# If the master table has a column named `region_id`,
# only the specified regions will be returned.
# Let us assume that we want to replicate case 3,
# but, now, we load only the first 20 regions.
master_table <- data.table(path=paths,
                           treatment=c("control", "drug_A"),
                           region_id=rep(1:20,each= 2))
# We could also imagine that every even region contains a male,
# whilst every odd one has a female:
master_table[, sex := ifelse(region_id %% 2, "male", "female" )]
# Note that we have now two conditions.
# Let us check our new table:
print(master_table)
# Then we can load our data:
dt <- loadPsvData(master_table)
# This is simply a subset of data, so many regions are missing
# lets display the regions we ended up with
print(dt[,.(NA),by=key(dt)])
#####
# Case 5: Apply ANALYSIS/function whilst loading the data.
# You can also apply a function from this package,
# or your own function to the data as it is being loaded.
# For instance, if you wish to perform a `sleep annotation`:
dt <- loadPsvData(paths[1], FUN=sleepAnnotation)
# You could of course combine this with more conditions/region selection.
# For most complicated cases, you would probably have pre-generated the
# master-table (e.g. as a csv file) before analysing the results.

```

loadSampleData

Retreive sample/example data contained within in this package.

Description

This function is only for testing (and trying) purposes. It provides a way to access raw data (e.g. db files) contained within this package.

Usage

```
loadSampleData(names = NULL)
```

Arguments

names The name of the samples to be loaded. When names is NULL, the function returns the list of all available samples.

See Also

[loadPsvData](#) to obtain raw experimental data.

maxVelocityClassifier *Motion classifier based on maximum velocity.*

Description

Defines whether an animal is moving according to its subpixel velocity. It requires a variable named xy_dist_log10x1000 in the .db file.

Usage

```
maxVelocityClassifier(data, velocity_threshold = 0.005)
```

Arguments

data the data.table containing behavioural features used for movement classification.
velocity_threshold velocity above which an animal is classified as ‘moving’.

See Also

[sleepAnnotation](#) to apply this function to all subsequent time windows.

mins *Trivially converts minutes to seconds*

Description

Trivially converts minutes to seconds

Usage

```
mins(x)
```

Arguments

x number of seconds

Value

the corresponding number of seconds

See Also

[days](#) [hours](#)

| | |
|--------------|--|
| overviewPlot | <i>Displays, per individual, the temporal average of a variable of interest.</i> |
|--------------|--|

Description

This function produces a tiled representation in which every row represents one individual (i.e. from a unique combination of region and experiment). The x axis represents time in days. The values of the variable of interest are represented by different colour intensities.

Usage

```
overviewPlot(y, data, condition = NULL, summary_time_window = mins(30),
             normalise_var_per_id = TRUE)
```

Arguments

| | |
|----------------------|---|
| y | The variable of interest |
| data | The data.table containing the data. It must have a column with the same name as y. |
| condition | An optionnal grouping factor to order rows. |
| summary_time_window | the width (in seconds) of the time window used to draw each pixel. |
| normalise_var_per_id | whether each row is to be normalised, using $\text{new_y} = (y - \text{mean}(y))/\text{sd}(y)$. |

Value

A ggplot object that can be plotted directly or modified.

See Also

[ethogramPlot](#) To show trend by aggregating individuals over time.

Examples

```
# Load sample data
data(monitor_validation)
# We perform sleep annotation on all animals
my_data <- monitor_validation[,sleepAnnotation(.SD),by=key(monitor_validation)]
# let us have a look of the max velocity as a measure of activity
p <- overviewPlot(max_velocity,my_data)
print(p)
# Let us make a dummy treatment variable.
# For instance, region > 10 have a treatment named ``Drug_1``
my_data[,treatment:=ifelse(region_id > 10,"Control","Drug_1")]
p <- overviewPlot(max_velocity,my_data,condition = treatment)
# See how treatment levels are grouped together:
print(p)
# p is simply a ggplot object, so we can change things:
print(p + labs(title="MY own title"))
```

| | |
|-----------------|---|
| sleepAnnotation | <i>Determines whether an animal is asleep</i> |
|-----------------|---|

Description

This function uses a motion classifier to first decide whether an animal is moving during a given time window. Then, it defines sleep as contiguous immobility for a minimal duration.

Usage

```
sleepAnnotation(data, time_window_length = 10, min_time_immobile = 60 * 5,
  motion_classifier_FUN = maxVelocityClassifier, ...)
```

Arguments

| | |
|-----------------------|--|
| data | the data (i.e a data.table) from a <i>single</i> region. It must contain, at least, the columns 't', 'x' and 'y'. |
| time_window_length | The number of seconds to be used by the motion classifier. This corresponds to the sampling period of the output data. |
| min_time_immobile | the minimal duration (in s) after which an immobile animal is scored as 'asleep'. |
| motion_classifier_FUN | the function used to classify movement. |
| ... | extra arguments to be passed to motion_classifier_FUN |

Value

A data table similar to data with additionnal variables/annotations (i.e. 'moving', 'asleep').

Note

The resulting data will only have one data point every time_window_length seconds.

See Also

[loadPsvData](#) to load data and optionnaly apply analysis on the fly.

Examples

```
# Let us load some sample data
data(tube_monitor_validation)
# We will start only with region 2:
dt_region2 <- tube_monitor_validation[region_id==2,]
sleep_dt <- sleepAnnotation(dt_region2)
print(sleep_dt)
# We make a sleep 'barcode'
ggplot(sleep_dt, aes(t,region_id,fill=asleep)) + geom_tile()
# A bit of data.table wizardry to apply that to each experieiment and region:
sleep_dt <- tube_monitor_validation[,sleepAnnotation(.SD),by=key(tube_monitor_validation)]
# The same bare code for all regions
ggplot(sleep_dt, aes(t,region_id,fill=asleep)) + geom_tile()
```

`virtualBeamCrossClassif`*Motion classifier based on beam crosses.*

Description

Defines whether an animal is moving. This is achieved by computing the number of crossed of a "virtual beam" in the middle of its region (i.e. at $x=0.5$). This emulate the type of data generated by DAM2.

Usage

```
virtualBeamCrossClassif(data)
```

Arguments

`data` the data.table containing behavioural features used for movement classification.

See Also

[maxVelocityClassifier](#) to define movement by maximum velocity, which is more accurate, instead.

Index

boutAnalysis, [2](#)

days, [2](#), [5](#), [10](#)

ethogramPlot, [3](#), [11](#)

fetchPsvResultFiles, [4](#)

hours, [2](#), [5](#), [10](#)

loadDAMFile, [5](#)

loadMetaData, [6](#), [6](#), [8](#)

loadPsvData, [4](#), [7](#), [7](#), [10](#), [12](#)

loadSampleData, [9](#)

maxVelocityClassifier, [10](#), [13](#)

mins, [2](#), [5](#), [10](#)

overviewPlot, [11](#)

oveviewPlot, [3](#)

sleepAnnotation, [10](#), [12](#)

virtualBeamCrossClassif, [13](#)