# Package 'rethomics'

June 18, 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,
reshape2

**LazyLoad** TRUE

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

**URL** https://github.com/gilestrolab/rethomics

## R topics documented:

---

`boutAnalysis` *Todo*

---

## Description

Todo

## Usage

```
boutAnalysis(var, data)
```

---

`days` *Trivially converts days to seconds*

---

## 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` *Displays the temporal and inter-individual average of a variable of interest.*

## 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 `new_x = (x - mean(x))/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

`oveviewPlot` 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"))
```

`fetchDAMData` *Retreives DAM2 data from data that is saved daily*

## Description

Uses a query mechanism to get data from a DAM2 array. This is useful when data has been saved, by day, in individual files for each monitor.

## Usage

```
fetchDAMData(result_dir, query, reference_hour = 9, tz = "BST")
```

## Arguments

| | |
|---|---|
| `result_dir` | the root directory where all daily data are saved |
| `query` | a formated query used to request data (see detail). |
| `reference_hour` | the hour, in the day, to use as t_0 reference. This should be expressed on Greenwich Meridian Time. |
| `tz` | the time zone on which the DAM2 data was saved (e.g. BSM -> British Summer Time) |

## Details

`query` must be a data.table. Conceptually, each row of the query describes the conditions in one channel (when `region_id` is specified), or in each monitor (when it is not). It should have the following columns:

- `machine_id` the name of the machine used (e.g. 'M002').
- `start_date` the first day of the requested experiement (e.g. '2014-12-28').
- `stop_date` the last day of the requested experiement (e.g. '2014-12-30').
- `region_id` the channel (between 1 and 32) in what the animal was in (e.g. '20'). This is an optionnal column. If not povided, all 32 channels are loaded with the same conditions.
- `...` arbitrary columns to assotiate conditions/treatments/genotypes/... to the previous columns

## Value

A data.table where every row is an individual measurment. That is an activity at a unique time (`t`) in a unique channel (`region_id`), and from a unique result date/experiment (`experiment_id`). The time is expressed in seconds. For each different combination of `start_date` and `machine_id` in the query, an individual `experiment_id` is generated.

## Note

the daily data should be saved in a hard-coded directory structure `root_dir/yyyy/mm/mmdd/mmddMxyz.txt`, where:

- `yyyy` Is the year (e.g. 2014)
- `mm` and `dd`, the formated month and day, respectively (e.g. mm=12 and dd=28).
- `xyz`, the number of the monitor (e.g 003)

`fetchPsvResultFiles` *Query files from a PSV data directory according to the date of the experiment and the device which acquired the data.*

## Description

This function is designed to list and select experiemental files. In general, end-users will want to retreive path to their experimental files according to the date and ID of the video monitor without having to undertand 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 formated 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 formated 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 happends, *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 *Trivially converts hours to seconds*

---

## 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 *Read a text file formated as DAM2 into a single data table.*

---

## Description

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

## Usage

```
loadDAMFile(FILE, tz = "")
```

### Arguments

| | |
|---|---|
| FILE | the name of the input file. |
| tz | the time zone of the computer saving the file. By default, tz is taken from the computer running this function |

### Value

a data table with an activity (number of beam crosses) variable, a region_id (channel) variable and a posix time stamp.

### 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 realistec example where we have experiemental 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 *Retreive 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– embeded 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 *Read data from a result file.*

---

## 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 experiement. |
| 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) immedidatly after is 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 additionnal conditions (i.e. data frame columns) to specific regions/files. When additionnal 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 measurment. 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 with 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 measurment originates from.
print(dt)

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

---

`makeBoutDt` *set.seed(1) bout_length <- round(runif(100,1,100)) bout_val <- rep(c(T,F),length.out=length(bout_length)) x <- rep(bout_val,bout_length) test_dt <- data.table(x=x, t=1:length(x) + 50) makeBoutDt(x,test_dt)*

---

## Description

set.seed(1) bout_length <- round(runif(100,1,100)) bout_val <- rep(c(T,F),length.out=length(bout_length)) x <- rep(bout_val,bout_length) test_dt <- data.table(x=x, t=1:length(x) + 50) makeBoutDt(x,test_dt)

## Usage

```
makeBoutDt(x, sub_data)
```

---

`maxVelocityClassifier` *Motion classifier based on maximum velocity.*

---

## Description

Defines whether an animal is moving according to its subpixel velocity. It requieres 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 *Displays, per individual, the temporal average of a variable of interest.*

### Description

This function produces a tiled representation in which every row represents one individual (i.e. from a unique combination of region and experiement). 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 `new_y = (y - mean(y))/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 *Determines whether an animal is asleep*

---

## 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 *single* 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 `barecode'
ggplot(sleep_dt, aes(t,region_id,fill=asleep)) + geom_tile()
# A bit of data.table wizardry to apply that to each experiement 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 defince movement by maximum velocity, which is more accurate, instead.

# Index