Package ‘ggetho’

April 29, 2020

Title Visualisation of High-Throughput Behavioural (i.e. Ethomics) Data

Date 2020-04-29

Version 0.3.6

Description Extension of 'ggplot2' providing layers, scales and preprocessing functions useful to represent behavioural variables that are recorded over multiple animals and days. This package is part of the 'rethomics' framework <http://rethomics.github.io/>.

Depends R (>= 3.00), ggplot2, behavr
Imports data.table, stringr, scales, labeling, rlang
Suggests testthat, covr, knitr, zeitgebr
License GPL-3
Encoding UTF-8
LazyData true

URL https://github.com/rethomics/ggetho

BugReports https://github.com/rethomics/ggetho/issues
RoxygenNote 6.1.1

NeedsCompilation no

Author Quentin Geissmann [aut, cre]
Maintainer Quentin Geissmann <qgeissmann@gmail.com>
Repository CRAN

Date/Publication 2020-04-29 19:30:02 UTC

R topics documented:

geom_peak .......................................................... 2
ggetho ............................................................... 4
ggperio .............................................................. 6
ggspectro ........................................................... 7
id_labeller .......................................................... 8
geom_peak

Description
This function draws points on the x-y coordinates of selected peaks and write their (y) value on the bottom of the plot.

Usage
geom_peak(mapping = NULL, data = NULL, stat = "identity", position = "identity", ..., na.rm = TRUE, show.legend = NA, inherit.aes = TRUE, peak_rank = 1, conversion = hours)

Arguments
mapping
Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data
The data to be displayed in this layer. There are three options:
If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot().
A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.
A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x,10)).
stat
The statistical transformation to use on the data for this layer, as a string.
position
Position adjustment, either as a string, or the result of a call to a position adjustment function.
... Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.
na.rm
If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend
logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
geom_peak

\[\text{inherit.aes}\]
If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().

\[\text{peak_rank}\]
numerical vector specifying the rank(s) of peak(s) to draw

\[\text{conversion}\]
function to convert values of \(x\) to a specific unit. The default, hours, will write \(x\) (time) in decimal hours.

Details
In the input data, peaks are encoded as an additional column/aesthetic with values corresponding to peak ranks (and 0 when the point is not a peak). In other word, the mapping must provide \(x\), \(y\) and peak. Only peaks matching peak_rank will be drawn (see example).

Value
A ggplot layer.

References
- The relevant rethomic tutorial section

See Also
- ggperio to create a periodogram
- zeitgebr::find_peaks to automatically add a peak column on periodogram data

Other layers: stat_bar_tile_etho, stat_ld_annotations, stat_pop_etho

Examples
```r
# We make a data frame by hand with five rows
# There are two peaks: in position 4 and 2

df <- data.frame(x = hours(1:5),
                 y = c(1, 2, 0, 4, 1),
                 peak = c(0, 2, 0, 1, 0))

# We draw the plot as a line
pl <- ggplot(df, aes(x, y, peak = peak)) +
     geom_line() +
     scale_x_hours()

pl

# Now we could add the peak values as an extra layer:
# The first peak
pl + geom_peak()
# The first and second peak
pl + geom_peak(peak_rank = 1:2)
# The second only
pl + geom_peak(peak_rank = 2)

# Just like with other geoms,
# we can change colour, size, alpha, shape, ...
```
## In the context of circadian analysis,

Using the zeitgebr package:

```r
require(zeitgebr)
# We make toy data
metadata <- data.table(id = sprintf("toy_experiment|%02d", 1:40),
                      region_id = 1:40,
                      condition = c("A", "B"),
                      sex = c("M", "M", "F", "F"))
dt <- toy_activity_data(metadata, seed = 107)
# We shift period of the group "A" by 0.01
dt[, t := ifelse(xmv(condition) == "A", t, t * 1.01)]
# We compute a periodogram for each individual
per_dt <- periodogram(moving, dt, FUN = chi_sq_periodogram)
per_dt <- find_peaks(per_dt)
out <- ggperio(per_dt, aes(y = power - signif_threshold, colour = condition, peak = peak)) +
      stat_pop_etho() +
      facet_wrap( ~ id, labeller = id_labeller)
out
out + geom_peak(colour="black")
```

---

**ggetho**

Prepare a ggplot object to represent behavioural data

**Description**

This function summarises a variable of interest (y or z axis) in order to subsequently represent it over time (x axis) (using layers provided either by ggplot2 or ggetho).

**Usage**

```r
ggetho(data, mapping, summary_FUN = mean,
       summary_time_window = mins(30), time_wrap = NULL, time_offset = 0,
       multiplot = NULL, multiplot_period = hours(24), ...)
```

**Arguments**

- **data**
  - behavr::behavr table containing the data and metadata
- **mapping**
  - default list of aesthetic mappings to use for plot
- **summary_FUN**
  - method (function) used to summarise variable over time (typically, the mean)
- **summary_time_window**
  - width (in seconds) of the time window to compute a summary on
- **time_wrap**
  - time (in seconds) used to wrap the data (see details)
- **time_offset**
  - time offset (i.e. phase, in seconds) when using time_wrap
multiplot integer, greater than two, or NULL, the default (see details)
multiplot_period

the duration of the period when multiplotting (see details)

... additional arguments to be passed to `ggplot2::ggplot()`

Details

time_wrap is typically used to express time relatively to the start of the day. In other words, it can help be used to pull all days together in one representative day. In this case, `time_wrap = hours(24)`. Instead of representing data from the start of the day, it can be done from any offset, using `time_offset`. For instance, `time_offset = hours(12)` puts the circadian reference (ZT0) in the middle of the plot.

Multiplotting is a generalisation of double-plotting, triple-plotting... This type of representation is useful to understand periodic behaviours. When `multiplot` is not NULL, data is repeated as many time as its value, along the x axis. The y axis is then the period (typically the day) onset. It is possible to set duration of the period, which is typically 24 h to arbitrary values using the `multiplot_period` argument.

Value

An initial ggplot object that can be further edited.

References

- The relevant rethomic tutorial section

See Also

- `stat_pop_etho` to show population trend by aggregating individuals over time
- `stat_tile_etho` to show variable of interest as colour intensity
- `stat_l_d_annotations` to show light and dark phases on the plot

Examples

```r
# We start by making a dataset with 20 animals
metadata <- data.table(id = sprintf("toy_experiment|%02d", 1:20),
                        condition = c("A", "B"))
dt <- toy_activity_data(metadata, 3)
# We build a plot object with **nothing inside** (just the axis)
# we want to show proportion of time sleeping on the y axis:
pl <- ggetho(dt, aes(y = asleep))
pl
# Sometimes, the variable of interest is not on the y axis, but on z axis (colour scale).
# When we do not provide a y axis,
# ggetho will make an ID fo each animal and display them on separate rows
pl <- ggetho(dt, aes(z = asleep))
pl
# this one is the same type, but it groups the animals by condition
pl <- ggetho(dt, aes(z = asleep, y = condition))
```
```
pl <- ggetho(dt, aes(z = asleep, y = paste(condition, id)))
pl

# we want to summarise (wrap) data along a circadian day:
pl <- ggetho(dt, aes(y = asleep), time_wrap = hours(24))
pl

# double-plotted actogram:
pl <- ggetho(dt, aes(z = moving),
            multiplot = 2,
            multiplot_period = hours(24))
pl

# then use `+ stat_tile_etho()`, or `+ stat_bar_tile_etho()`
```

---

**ggperio**

Prepare a ggplot object to represent periodogram data

**Description**

This function summarises periodogram data (containing periodograms of multiple individual), to show period on the x axis, and power (or equivalent) on the y axis.

**Usage**

```
ggperio(data, mapping = aes(x = period, y = power), ...)
```

**Arguments**

- `data` : behavr::behavr table containing the data and metadata
- `mapping` : default list of aesthetic mappings to use for plot
- `...` : additional arguments to be passed to `ggplot2::ggplot()`

**Value**

An initial ggplot object that can be further edited.

**References**

- The relevant rethomic tutorial section

**See Also**

- `ggetho` to plot time series
- `geom_peak` to draw peaks on a periodogram
- `zeitgebr::periodogram` to compute periodograms in a first place
- `ggspectro` to visualise spectrograms
Examples

```r
require(zeitgebr)
# We make toy data
metadata <- data.table(id = sprintf("toy_experiment|%02d", 1:40),
                      region_id = 1:40,
                      condition = c("A", "B"),
                      sex = c("M", "M", "F", "F"))
dt <- toy_activity_data(metadata, seed = 107)
# We shift period of the group "A" by 0.01
dt[, t := ifelse(xmv(condition) == "A", t, t * 1.01)]
# We compute a periodogram for each individual
per_dt <- periodogram(moving, dt, FUN = chi_sq_periodogram)

# Then we display them as an average
out <- ggperio(per_dt, aes(y = power, colour = condition))
out + stat_pop_etho()

out <- ggperio(per_dt, aes(y = power - signif_threshold, colour = condition))
out + stat_pop_etho()
out <- ggperio(per_dt, aes(y = power - signif_threshold, colour = condition))
out + stat_pop_etho() + facet_wrap(~ id, labeller = id_labeller)
```

---

**ggspectro**

Prepare a ggplot object to represent spectrogram data

**Description**

This function summarises spectrogram data (containing spectrograms of multiple individual), to show period on the y axis, time on the x axis and power on the z axis (e.g. as a colour).

**Usage**

```r
ggspectro(data, mapping = aes(), summary_FUN = mean,
          summary_time_window = mins(30), time_wrap = NULL, time_offset = 0,
          ...)
```

**Arguments**

- `data` - `behavr::behavr` table containing the data and metadata
- `mapping` - default list of aesthetic mappings to use for plot
- `summary_FUN` - method (function) used to summarise variable over time (typically, the mean)
- `summary_time_window` - width (in seconds) of the time window to compute a summary on
- `time_wrap` - time (in seconds) used to wrap the data (see details)
- `time_offset` - time offset (i.e. phase, in seconds) when using `time_wrap`
- `...` - additional arguments to be passed to `ggplot2::ggplot()`
Value

An initial ggplot object that can be further edited.

References

• The relevant rethomic tutorial section

See Also

• ggperio to visualise periodograms

Examples

```r
library(zeitgebr)
data(dams_sample)
dt <- dams_sample
spect_dt <- spectrogram(activity, dt)
pl <- ggspectro(spect_dt, time_wrap = hours(24)) +
  stat_tile_etho() +
  scale_y_hours(log=TRUE) +
  stat_ld_annotations(ld_colours = c("grey","black"))
pl + facet_grid(period_group ~ .)
pl + facet_wrap(~ id)
```

---

id_labeller

A facet labeller for id

Description

This function returns a ggplot2::labeller that displays the id on several lines to improve readability.

Usage

```r
id_labeller(labels)
```

Arguments

labels Data frame of labels. Usually contains only one element, but faceting over multiple factors entails multiple label variables.

See Also

ggplot2::labeller, to make your own labellers
Examples

```r
library(behavr)
metadata <- data.frame(
  id = sprintf("2017-09-01 20:00:12|toy_experiment_a_very_long_name|%02d", 1:20),
  condition = c("A", "B"))
dt <- toy_activity_data(metadata, duration = hours(2))
pl <- ggetho(dt, aes(y = asleep)) + stat_pop_etho()
## Without labelling
pl + facet_wrap(~ id)
## With labeller
pl + facet_wrap(~ id, labeller = id_labeller)
```

**stat_bar_tile_etho**

Display a variable of interest either as a colour intensity value or as a bar height

Description

These functions show the temporal trend (time on the x axis) of a variable of interest (z axis) as either colour intensity (`stat_tile_etho`) or using the height of the tiles (`stat_bar_tile_etho`). In both cases, the y axis is a discrete variable such as a treatment or the id of individuals.

Usage

```r
stat_bar_tile_etho(mapping = NULL, data = NULL, geom = "bar_tile",
  position = "identity", ..., method = mean, method.args = list(),
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
stat_tile_etho(mapping = NULL, data = NULL, geom = "raster",
  position = "identity", ..., method = mean, method.args = list(),
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

Arguments

- **mapping**
  Set of aesthetic mappings created by `aes()` or `aes_()`. If specified and `inherit.aes` = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

- **data**
  The data to be displayed in this layer. There are three options:
  - If NULL, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x,10)`).
geom** The geometric object to use to display the data.**

**position** Position adjustment, either as a string, or the result of a call to a position adjustment function.

**...** Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

**method** Function used to compute the aggregate, when grouping several individuals on the same row. The default is function is `mean`, `median`, `min`, `max` are examples of alternatives.

**method.args** List of additional arguments passed on to the modelling function defined by `method`.

**na.rm** If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

**show.legend** Logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

**inherit.aes** If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.

**Value**

A ggplot layer.

- The relevant rethomic tutorial section

**See Also**

- `ggeitho` to generate a plot object
- `stat_pop_etho` to show population trend by aggregating individuals over time
- `stat_ld_annotations` to show light and dark phases on the plot

Other layers: `geom_peak`, `stat_ld_annotations`, `stat_pop_etho`

**Examples**

```r
# We start by making a dataset with 20 animals
metadata <- data.frame(id = sprintf("toy_experiment | %02d", 1:20),
                         age = c(1, 5, 10, 20),
                         condition = c("A", "B"))
print(metadata)
dt <- toy_activity_data(metadata, 3)
# We build a plot object
pl <- ggetho(dt, aes(z = asleep))
# A standard plot one row per animal:
pl + stat_tile_etho()
# We can also group animals per condition and calculate the average sleep
pl <- ggetho(dt, aes(z = asleep, y = condition))
```

---
pl + stat_tile_etho()

# We can sort by adding condition AND id on the y axis:
pl <- ggetho(dt, aes(z = asleep, y = interaction(id, condition)))
pl + stat_tile_etho()

# Same if we want to sort by age
pl <- ggetho(dt, aes(z = asleep, y = interaction(id, age)))
pl + stat_tile_etho()

# Instead of the average, maybe we want to show the highest (max)
# possible value of sleep for any time point
pl + stat_tile_etho(method = max)

# We can also use stat_bar_tile as an alternative
pl + stat_bar_tile_etho()

---

**stat_ld_annotations**

*Compute and display light/dark annotations onto a plot object*

**Description**

This function is used to show light and dark (L and D) phases as boxes on top a plot.

**Usage**

```r
stat_ld_annotations(mapping = NULL, data = NULL,
  position = "identity", ld_colours = c("white", "black"),
  ypos = "bottom", height = 0.03, period = hours(24), phase = 0,
  l_duration = hours(12), outline = "black", x_limits = c(NA, NA),
  ..., na.rm = FALSE, show.legend = FALSE, inherit.aes = TRUE)
```

**Arguments**

- **mapping**
  Set of aesthetic mappings created by `aes()` or `aes()`. If specified and `inherit.aes` = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

- **data**
  The data to be displayed in this layer. There are three options:
  - If NULL, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(., 10)`).

- **position**
  Position adjustment, either as a string, or the result of a call to a position adjustment function.
ld_colours  character vector of length two setting the colours for light and dark phases, respectively. The default is c("white", "black").

ypos  position and height of the annotation on the y axis. It can be either "top" or "bottom". The default, "bottom" will put the labels below any data.

height  relative height of the rectangles. The default is 3 percent (0.03).

period, phase, l_duration  period, phase and duration of the L phase (in seconds) of the LD cycle.

outline  colour of the border of the rectangles. A value of NA draws no border.

x_limits  numerical vector of length 2 for the start and end of the annotations (in seconds). The default, c(NA, NA), uses the full range of the plotted data.

...  Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

na.rm  If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend  logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes  If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().

Value
A ggplot layer.

References
• The relevant rethomic tutorial section

See Also
• ggetho to generate a plot object

Other layers: geom_peak, stat_bar_tile_etho, stat_pop_etho

Examples
library(behavr)
# We start by making a to dataset with 20 animals
metadata <- data.frame(id = sprintf("toy_experiment | %02d", 1:20),
    condition = c("A", "B"))
dt <- toy_activity_data(metadata, 3)
# We build a plot object
pl <- ggetho(dt, aes(y = asleep)) + stat_pop_etho()
pl + stat_ld_annotations()
# We can also put the annotations in the background:
pl <- ggetho(dt, aes(y = asleep)) +
stat_pop_etho

```
stat_ld_annotations(outline = NA) +
stat_pop_etho()
pl
# Different colours (e.g. DD)
pl + stat_ld_annotations(ld_colour = c("grey", "black"))
# Shorter period
pl + stat_ld_annotations(period = hours(22), phase = hours(3))
# On a tile plot:
pl <- ggetho(dt, aes(z = asleep)) + stat_tile_etho()
pl + stat_ld_annotations()
```

---

**stat_pop_etho**  
*Compute and display a population aggregate for a variable of interest*

### Description

This function displays the temporal (time on the x axis) trend of variable of interest, on the y axis as a line with confidence interval as a shaded area.

### Usage

```
stat_pop_etho(mapping = NULL, data = NULL, geom = "smooth",
          position = "identity", ..., method = mean_se, method.args = list(),
          show.legend = NA, inherit.aes = TRUE)
```

### Arguments

- **mapping**  
  Set of aesthetic mappings created by `aes()` or `aes_()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

- **data**  
  The data to be displayed in this layer. There are three options:
  - If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x,10)`).

- **geom**  
  The geometric object to use display the data

- **position**  
  Position adjustment, either as a string, or the result of a call to a position adjustment function.

- **...**  
  Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.
method function used to compute the aggregate and confidence intervals. It should return \((y, ymin \text{ and } ymax)\). The default is `ggplot2::mean_se`, which computes the mean + or - standard error. `ggplot2::mean_cl_boot` can be used instead to generate bootstrap confidence interval instead.

method.args List of additional arguments passed on to the modelling function defined by method.

show.legend logical. Should this layer be included in the legends? \(\text{NA}\), the default, includes if any aesthetics are mapped. \(\text{FALSE}\) never includes, and \(\text{TRUE}\) always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes If \(\text{FALSE}\), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.

Value

A ggplot layer.

References

• The relevant rethomic tutorial section

See Also

• `ggetho` to generate a plot object
• `stat_tile_etho` to show variable of interest as colour intensity
• `stat_ld_annotations` to show light and dark phases on the plot
• `ggplot2::stat_smooth` to understand how to change the type of confidence interval, line colour and so forth

Other layers: `geom_peak, stat_bar_tile_etho, stat_ld_annotations`

Examples

```r
library(behavr)
metadata <- data.frame(id = sprintf("toy_experiment | %02d", 1:20),
  age=c(1, 5, 10, 20),
  condition=c("A", "B"))

dt <- toy_activity_data(metadata, 3)
# We build a plot object
pl <- ggetho(dt, aes(y = asleep))
# A standard plot of the whole population:
pl + stat_pop_etho()
# We can also split by condition, and display the two population on different facets:
pl + stat_pop_etho() + facet_grid(condition ~ .)

# Instead, we can use different colour for separate conditions:
pl <- ggetho(dt, aes(y = asleep, colour = condition))
pl + stat_pop_etho()

# Sometimes, we also have numeric condition (e.g. age)
```
```r
pl <- ggetho(dt, aes(y = asleep, colour = age))
pl + stat_pop_etho()
# We could want to aggregate several days of data to one circadian day (i.e. time wrapping)
# here, we also plot the invert of moving (!moving)
pl <- ggetho(dt, aes(y = !moving), time_wrap = hours(24))
pl + stat_pop_etho()
```

---

### Description

A set of scales used to represent experimental durations.

### Usage

```r
scale_x_days(name = "Time", breaks = waiver(),
             minor_breaks = waiver(), labels = waiver(), limits = NULL,
             expand = waiver(), oob = scales::censor, na.value = NA_real_,
             position = "bottom", time_wrap = NULL, unit = "day", log = FALSE)

day <- scale_x_days()

scale_y_days(name = "Time", breaks = waiver(),
             minor_breaks = waiver(), labels = waiver(), limits = NULL,
             expand = waiver(), oob = scales::censor, na.value = NA_real_,
             position = "left", time_wrap = NULL, unit = "day", log = FALSE)

scale_x_hours(name = "Time", breaks = waiver(),
              minor_breaks = waiver(), labels = waiver(), limits = NULL,
              expand = waiver(), oob = scales::censor, na.value = NA_real_,
              position = "bottom", time_wrap = NULL, unit = "h", log = FALSE)

scale_y_hours(name = "Time", breaks = waiver(),
              minor_breaks = waiver(), labels = waiver(), limits = NULL,
              expand = waiver(), oob = scales::censor, na.value = NA_real_,
              position = "left", time_wrap = NULL, unit = "h", log = FALSE)

scale_x_seconds(name = "Time", breaks = waiver(),
                 minor_breaks = waiver(), labels = waiver(), limits = NULL,
                 expand = waiver(), oob = scales::censor, na.value = NA_real_,
                 position = "bottom", time_wrap = NULL, unit = "s", log = FALSE)

scale_y_seconds(name = "Time", breaks = waiver(),
                 minor_breaks = waiver(), labels = waiver(), limits = NULL,
                 expand = waiver(), oob = scales::censor, na.value = NA_real_,
                 position = "left", time_wrap = NULL, unit = "s", log = FALSE)
```
Arguments

name
The name of the scale. Used as the axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.
breaks
One of:
• waiver() for no breaks
• A Date/POSIXct vector giving positions of breaks
• A function that takes the limits as input and returns breaks as output

minor_breaks
One of:
• waiver() for no breaks
• A Date/POSIXct vector giving positions of minor breaks
• A function that takes the limits as input and returns minor breaks as output

labels
One of:
• waiver() for the default labels computed by the transformation object
• A character vector giving labels (must be same length as breaks)
• A function that takes the breaks as input and returns labels as output

limits
One of:
• waiver() for the default scale range
• A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum
• A function that accepts the existing (automatic) limits and returns new limits

expand
Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function ggplot2::expand_scale() to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.
oob
Function that handles limits outside of the scale limits (out of bounds). The default replaces out of bounds values with NA.
na.value
Missing values will be replaced with this value.
position
The position of the axis. "left" or "right" for vertical scales, "top" or "bottom" for horizontal scales
time_wrap
duration (in seconds) used to wrap the labels of the time axis
unit
the name of unit (string) to be used in the label (e.g. one could use "second" instead of "s")
log
logical, whether axis should be on a log-transformed

Details
time_wrap is useful, for instance, to express time within a day (ZT), instead of absolute time.
time_scales

Value

A ggplot scale.

References

- The relevant rethomic tutorial section

See Also

- `ggetho` to generate a plot object
- `ggplot2::scale_x_continuous`, the default ggplot scale, to understand limits, breaks, labels and name

Examples

```r
# We generate some data
metadata <- data.frame(id = sprintf("toy_experiment | %02d", 1:20),
    condition = c("A", "B"))
dt <- toy_activity_data(metadata, 3)
# Then, a simple plot
pl <- ggetho(dt, aes(y = asleep)) + stat_pop_etho()
pl + scale_x_hours(breaks = days(c(1, 2)))
pl + scale_x_hours()
pl + scale_x_days(breaks = days(c(1, 2)))
pl + scale_x_days()

# To express time modulus `time_wrap`
# e.g. time in the day
pl + scale_x_hours(time_wrap = hours(24)) +
    coord_cartesian(xlim=c(0, days(2)))

# On a shorter time scale
pl <- ggetho(dt[t < hours(5)], aes(z = asleep)) + stat_tile_etho()
pl + scale_x_hours()
pl + scale_x_hours(breaks = hours(1:4))
pl + scale_x_seconds(breaks = hours(1:4))
```
Index

aes(), 2, 9, 11, 13
aes_(), 2, 9, 11, 13

behavr::behavr, 4, 6, 7
borders(), 3, 10, 12, 14

fortify(), 2, 9, 11, 13

ggplot2::ggplot(), 5–7
ggplot2::labeller, 8
ggplot2::mean_cl_boot, 14
ggplot2::mean_se, 14
ggplot2::scale_x_continuous, 17
ggplot2::stat_smooth, 14
ggspectro, 6, 7

id_labeller, 8

layer(), 2, 10, 12, 13

max, 10
mean, 10
median, 10
min, 10

scale_x_days(time_scales), 15
scale_x_hours(time_scales), 15
scale_x_seconds(time_scales), 15
scale_y_days(time_scales), 15
scale_y_hours(time_scales), 15
scale_y_seconds(time_scales), 15
stat_bar_tile_etho, 3, 9, 12, 14
stat_ld_annotations, 3, 5, 10, 11, 14
stat_pop_etho, 3, 5, 10, 12, 13
stat_tile_etho, 5, 14
stat_tile_etho(stat_bar_tile_etho), 9

time_scales, 15
zeitgebr::find_peaks, 3
zeitgebr::periodogram, 6