Package: spiro (via r-universe)

July 12, 2024

Title Manage Data from Cardiopulmonary Exercise Testing

Version 0.2.1.9000

Description Import, process, summarize and visualize raw data from metabolic carts. See Robergs, Dwyer, and Astorino (2010) <doi:10.2165/11319670-000000000-00000> for more details on data processing.

License MIT + file LICENSE

URL https://github.com/ropensci/spiro, https://docs.ropensci.org/spiro/

BugReports https://github.com/ropensci/spiro/issues

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Imports ggplot2, xml2, readxl, knitr, cowplot, digest, signal

Suggests testthat (>= 3.0.0), vdiffr, rmarkdown, ggborderline

VignetteBuilder knitr

Config/testthat/edition 3

Repository https://ropensci.r-universe.dev

RemoteUrl https://github.com/ropensci/spiro

RemoteRef main

RemoteSha ab65b426440a91268a7e52a1a1247d262bb0eb6a

Contents

`add_bodymass` *Calculate additional variables related to body mass for cardiopulmonary exercise testing data*

Description

`add_bodymass()` adds body mass-related variables to processed gas exchange data.

Usage

```
add_bodymass(data, bodymass = NULL)
```

Arguments

`data`	A `data.frame` of the class `spiro` containing the gas exchange data. Usually the output of a `spiro` call.
`bodymass`	A numeric value to manually set the participant's body mass. Defaults to NULL to use body mass data from the file's meta data. Set to NA to ignore the meta data without setting a new body mass.

Details

Based on an individual's body mass, relative oxygen uptake (VO2_rel) and carbon dioxide output (VCO2_rel) are calculated. For running protocols, running economy (RE) is calculated.

Value

A `data.frame` of the class `spiro` containing the cardiopulmonary exercise testing data including variables related to body mass.

Examples

```
# get example file
file <- spiro_example("zan_gxt")

s <- spiro(file)
out <- add_bodymass(s, bodymass = 65.3)
head(out)
```

---

add_hr *Import and add heart rate data to cardiopulmonary exercise testing data*

Description

`add_hr()` imports an external file containing heart rate data and adds it to an existing gas exchange data file.

Usage

```
add_hr(data, hr_file, hr_offset = 0)
```

Arguments

`data`	A `data.frame` of the class `spiro` containing the gas exchange data. Usually the output of a `spiro` call.
`hr_file`	The absolute or relative path of a `*tcx` file that contains additional heart rate data.
`hr_offset`	An integer, corresponding to the temporal offset of the heart-rate file. By default the start of the heart rate measurement is linked to the start of the gas exchange measurement. A positive value means, that the heart rate measurement started after the begin of the gas exchange measurements; a negative value means it started before.

Details

Heart rate data will be imported from a `.tcx` file. After interpolating the data to full seconds, it is then matched to the imported gas exchange data.

Value

A `data.frame` of the class `spiro` containing the cardiopulmonary exercise testing data including heart rate data.

Examples

```
# Get example data
oxy_file <- spiro_example("zan_ramp")
hr_file <- spiro_example("hr_ramp.tcx")

# Import and process spiro data
oxy_data <- spiro(oxy_file)

# Add heart rate data
out <- add_hr(oxy_data, hr_file)
head(out)
```

`add_protocol` *Add a test protocol to an exercise testing data set*

Description

`add_protocol()` adds a predefined test protocol to an existing set of data from an exercise test.

Usage

```
add_protocol(data, protocol)
```

Arguments

`data`	A spiro `data.frame` containing the exercise testing data.
`protocol`	A `data.frame` containing the test protocol, as created by `set_protocol`, `set_protocol_manual` or `get_protocol`.

Value

A `data.frame` of the class `spiro` with cardiopulmonary parameters and the corresponding load data.

See Also

set_protocol for protocol setting with helper functions.

set_protocol_manual for manual protocol design.

get_protocol For automated extraction of protocols from raw data.

Examples

```
# Get example data
file <- spiro_example("zan_gxt")

s <- spiro(file)
out <- add_protocol(
  s,
```

```
  set_protocol(pt_pre(60), pt_steps(300, 50, 50, 7, 30))
)
head(out)
```

---

bw_filter *Smooth data with a (zero-phase) Butterworth filter*

Description

Internal function for spiro_smooth.

Usage

```
bw_filter(x, n = 3, W = 0.04, zero_lag = TRUE)
```

Arguments

x	A numeric vector on which the digital filter should be applied
n	Order of the Butterworth filter, defaults to 3
W	Low-pass cut-off frequency of the filter, defaults to 0.04
zero_lag	Whether a zero phase (forwards-backwards) filter should be applied.

Details

Digital filtering might be a preferable processing strategy for smoothing data from gas exchange measures when compared to moving averages. Robergs et al. (2010) proposes a third order Butterworth filter with a low-pass cut-off frequency of 0.04 for filtering VO2 data.

It should be noted that Butterworth filter comprise a time lag. A method to create a time series with zero lag is to subsequently apply two Butterworth filters in forward and reverse direction (forwards-backwards filtering). While this procedure removes any time lag it changes the magnitude of the filtering response, i.e. the resulting filter has not the same properties (order and cut-off frequency) as a single filter.

Value

A numeric vector of the same length as x.

Examples

```
# Get VO2 data from example file
vo2_vector <- spiro(spiro_example("zan_gxt"))$VO2

out <- bw_filter(vo2_vector)
head(out, n = 20)
```

get_anonid *Get the anonymization id from personal data*

Description

`get_anonid()` returns the anonymization id corresponding to given personal data.

Usage

```
get_anonid(name, surname, birthday = NULL)
```

Arguments

`name`	A character string, containing the participant's name as present in the raw data file.
`surname`	A character string, containing the participant's surname as present in the raw data file.
`birthday`	A character string, containing the participant's birthday as present in the raw data file. If no birthday data is available in the raw data, this is ignored.

Details

By default, the spiro package anonymizes personal information obtained from file meta data. The data are saved to the "info" attribute of a spiro() call. The default anonymization ensures that no personal information is accidentally revealed, e.g. by sharing spiro outputs as .Rda files.

While there is no way to directly deanonymize the data, get_anonid() allows you to recreate the ids, when meta data (name, surname and birthday) are known. Birthday is only used within the id generation if available in the original raw data.

To disable the anonymization process during import use `spiro(anonymize = FALSE)`

Value

A character string, containing the anonymized id.

Examples

```
get_anonid("Jesse", "Owens", "12.09.1913")
```

get_protocol *Guess a test protocol from a corresponding exercise testing data set*

Description

`get_protocol()` gets the underlying test protocol based on given load data.

Usage

```
get_protocol(data)
```

Arguments

`data`	A `data.frame` containing the exercise testing data. It is highly recommend to parse non-interpolated breath-by-breath data or processed data with a very short interpolating/averaging interval.

Value

A `data.frame` with the duration and load of each protocol step.

Examples

```
# Import example data
raw_data <- spiro_raw(data = spiro_example("zan_gxt"))

get_protocol(raw_data)
```

knit_print.spiro *Printing spiro data frames in a knitr context*

Description

`knit_print.spiro()` provides a method for printing `data.frames` from `spiro` within `knitr`.

Usage

```
## S3 method for class 'spiro'
knit_print(x, min = 10, max = 20, digits = 2, ...)
```

Arguments

`x`	A `data.frame` of the class `spiro` to be printed.
`min`	An integer, which sets the number of rows to which `x` will be limited in printing if row number exceed `max`.
`max`	An integer, setting the maximal number of rows to be not cut to `min` in printing.
`digits`	An integer giving the number of decimals to be rounded to.
`...`	Passing of additional arguments to `knit_print.default()`.

Details

Cardiopulmonary exercise testing data imported by spiro will often come in large data.frames. When knitting R Markdown documents these will normally be printed in full size.

This function provides a method for data.frames of the class spiro to limit the number of rows displayed to min if it exceeds max. The number of hidden data rows will be printed below the data.frame.

Value

The function prints its argument and returns it invisibly.

Examples

```
# Get example data
s <- spiro(spiro_example("zan_gxt"))

knitr::knit_print(s)
```

---

print.spiro *Printing spiro data frames*

Description

Printing method for spiro objects that rounds output to two decimals.

Usage

```
## S3 method for class 'spiro'
print(x, round = 2, ...)
```

Arguments

x	A data.frame of the class spiro to be printed.
round	An integer giving the number of decimals to be rounded to.
...	Passing of additional arguments to print.data.frame().

Value

The function prints its argument and returns it invisibly.

Examples

```
# Get example data
s <- spiro(spiro_example("zan_gxt"))

out <- print(s)
head(out)
```

`set_protocol` *Setting an exercise testing profile*

Description

`set_protocol()` allows to set a load profile for an exercise test based on profile sections.

Usage

```
set_protocol(...)

pt_pre(duration)

pt_wu(duration, load, rest.duration = 0)

pt_steps(
  duration,
  load,
  increment,
  count,
  rest.duration = 0,
  last.duration = NULL
)

pt_const(duration, load, count, rest.duration = 0, last.duration = NULL)
```

Arguments

`...`	Functions related to sections of the load profile, such as `pt_pre`, `pt_wu`, `pt_const` or `pt_step`. Sections will be evaluated in the order they are entered.
`duration`	A number, giving the duration of the test section or a single load within the test section (in seconds).
`load`	A number, giving the (initial) load of a section.
`rest.duration`	A number, specifying the duration of (each) rest (in seconds).
`increment`	A number, giving the difference in load between the current and the following load step.
`count`	An integer for the number of load sections.
`last.duration`	A number, giving the duration of the last load step (in seconds).

Value

A `data.frame` with the duration and load of each protocol step.

Functions

- `pt_pre()`: Add pre-measures to a load protocol
- `pt_wu()`: Add a warm up to a load protocol
- `pt_steps()`: Add a stepwise load protocol
- `pt_const()`: Add a constant load protocol

See Also

set_protocol_manual for manual protocol design.

get_protocol for automated extracting of protocols from raw data.

Examples

```
set_protocol(pt_pre(60), pt_wu(300, 100), pt_steps(180, 150, 25, 8, 30))
```

`set_protocol_manual` *Manually setting a testing profile*

Description

`set_protocol_manual()` allows to set any user-defined load profile for an exercise test.

Usage

```
set_protocol_manual(duration, load = NULL)

## Default S3 method:
set_protocol_manual(duration, load)

## S3 method for class 'data.frame'
set_protocol_manual(duration, load = NULL)
```

Arguments

`duration`	Either a numeric vector containing the duration (in seconds) of each load step, or a `data.frame` containing columns for duration and load.
`load`	A numeric vector of the same length as `duration` containing the corresponding load of each step. Not needed, if load and duration are both given in a `data.frame` as the first argument of the function.

Value

A `data.frame` with the duration and load of each protocol step.

Methods (by class)

- `set_protocol_manual(default)`: Default method when duration and load are given separately
- `set_protocol_manual(data.frame)`: Method for data frames with a duration and a load column

See Also

set_protocol for protocol setting with helper functions.

get_protocol For automated extracting of protocols from raw data.

Examples

```
set_protocol_manual(
  duration = c(300, 120, 300, 60, 300),
  load = c(3, 5, 3, 6, 3)
)

# using a data.frame as input
pt_data <- data.frame(
  duration = c(180, 150, 120, 90, 60, 30),
  load = c(200, 250, 300, 350, 400, 450)
)
set_protocol_manual(pt_data)
```

---

spiro *Import and process raw data from metabolic carts/spiroergometric measures*

Description

`spiro()` wraps multiple functions to import and process raw data from metabolic carts into a `data.frame`.

Usage

```
spiro(
  file,
  device = NULL,
  bodymass = NULL,
  hr_file = NULL,
  hr_offset = 0,
  protocol = NULL,
  anonymize = TRUE
)
```

Arguments

file	The absolute or relative path of the file that contains the gas exchange data.
device	A character string, specifying the device for measurement. By default the device type is guessed by the characteristics of the file. This can be overridden by setting the argument to "cortex", "cosmed", "vyntus" or "zan".
bodymass	Numeric value for the individual's body mass, if the default value saved in the file should be overridden.
hr_file	The absolute or relative path of a *tcx file that contains additional heart rate data.
hr_offset	An integer, corresponding to the temporal offset of the heart-rate file. By default the start of the heart rate measurement is linked to the start of the gas exchange measurement. A positive value means, that the heart rate measurement started after the begin of the gas exchange measurements; a negative value means it started before.
protocol	A data.frame by set_protocol or set_protocol_manual containing the test protocol. This is automatically guessed by default. Set to NA to skip protocol guessing.
anonymize	Whether meta data should be anonymized during import. Defaults to TRUE. See get_anonid for more information.

Details

This function performs multiple operations on raw data from metabolic carts. It imports the raw data from a file, which might be complemented by an additional .tcx file with heart rate data.

After using this function, you may summarize the resulting data frame with spiro_summary and spiro_max, or plot it with spiro_plot.

Value

A data.frame of the class spiro with cardiopulmonary parameters interpolated to seconds and the corresponding load data.

The attribute "protocol" provides additional information on the underlying testing protocol. The attribute "info" contains additional meta data from the original raw data file. The attribute "raw" gives the imported raw data (without interpolation, similar to calling spiro_raw).

Import

Different metabolic carts yield different output formats for their data. By default, this function will guess the used device based on the characteristics of the given file. This behavior can be overridden by explicitly stating the device argument.

The currently supported metabolic carts are:

- **CORTEX** (.xlsx, .xls or files .xml in English or German language)
- **COSMED** (.xlsx or .xls files, in English or German language)
- **Vyntus** (.txt files in French, German or Norwegian language)

- **ZAN** (`.dat` files in German language, usually with names in the form of `"EXEDxxx"`)

The spiro function can import personal meta data (name, sex, birthday, ...). By default this data is anonymized with `anonymize = TRUE`, see `get_anonid` for more information.

Processing

Breath-by-breath data is linearly interpolated to get data points for every full second. Based on the given load data, the underlying exercise protocol is guessed and applied to the data. If no load data is available or the protocol guess turns wrong, you can manually specify the exercise `protocol` by using `set_protocol` or `set_protocol_manual`. If you want to skip the automated protocol guessing without providing an alternative, set `protocol = NA`. Note that in this case, some functions relying on load data (such as `spiro_summary`) will not work.

Additional variables of gas exchange are calculated for further analysis. Per default the body mass saved in the file's metadata is used for calculating relative measures. It is possible to specify `bodymass` manually to the function, overriding that value.

Protocols, heart rate data and body mass information can also be given in a piping coding style using the functions `add_protocol`, `add_hr` and `add_bodymass` (see examples).

Examples

```
# get example file
file <- spiro_example("zan_gxt")

out <- spiro(file)
head(out)

# import with user-defined test profile
p <- set_protocol(pt_pre(60), pt_steps(300, 2, 0.4, 9, 30))
out2 <- spiro(file, protocol = p)
head(out2)

# import with additional heart rate data
oxy_file <- spiro_example("zan_ramp")
hr_file <- spiro_example("hr_ramp.tcx")

out3 <- spiro(oxy_file, hr_file = hr_file)
head(out3)

# use the add_* functions in a pipe
# Note: base R pipe requires R version 4.1 or greater)
## Not run:
spiro(file) |>
  add_hr(hr_file = hr_file, hr_offset = 0) |>
  add_bodymass(68.2)

## End(Not run)
```

spiro_example *Get path to spiro example*

Description

spiro_example returns the file path for example data files within the spiro package.

Usage

```
spiro_example(file = NULL)
```

Arguments

file	Name of the file, either "zan_gxt", "zan_ramp" or "hr_ramp.tcx". Leave the argument empty to get a vector with the paths of all three example files.

Value

A character vector with the absolute file path of the example file(s).

Examples

```
# get path of a specific example data file
spiro_example("zan_gxt")

# get all paths of example data files
spiro_example()
```

spiro_import *Import raw data from spiroergometric devices (deprecated)*

Description

This function has been deprecated as of package version 0.2.0. It will be removed in the next version release. Please use spiro for automated import and processing or spiro_raw to import only raw data.

Usage

```
spiro_import(file, device = NULL, anonymize = TRUE)
```

Arguments

file	The absolute or relative path of the file that contains the gas exchange data.
device	A character string, specifying the device for measurement. By default the device type is guessed by the characteristics of the file. This can be overridden by setting the argument to "cortex", "cosmed", "vyntus" or "zan".
anonymize	Whether meta data should be anonymized during import. Defaults to TRUE. See get_anonid for more information.

---

spiro_max *Return maximum values from cardiopulmonary exercise tests*

Description

spiro_max() returns a data.frame with the maximum gas exchange parameters of an exercise test.

Usage

```
spiro_max(data, smooth = 30, hr_smooth = FALSE)
```

Arguments

data	A data.frame of the class spiro containing the gas exchange data. Usually the output of a spiro call.
smooth	Parameter giving the filter methods for smoothing the data. Default is 30 for a 30-second moving average. "20b" will apply a 20-breath averaging for example. See spiro_smooth for further details and filter methods (e.g. Butterworth filters).
hr_smooth	A logical, whether smoothing should also apply to heart rate data. Default is FALSE, which means that the absolute maximum heart rate value is taken without smoothing.

Details

Before calculating the maximum values, the raw data is smoothed. Default smoothing method is a 30-second rolling average. See the smooth argument in spiro_smooth for more options, such as breath-based averages or digital filtering.

Parameters calculated are the maxima of oxygen uptake (absolute and relative), carbon dioxide output, minute ventilation, respiratory exchange ratio (RER), and heart rate. The maximum values are defined as the highest single data values after the smoothing.

For the maximum RER a different algorithm is used, as the RER during and after rest may exceed the peak value during exercise. Therefore only values during the last ten percent of the exercise time are considered for the RERmax determination. The RERmax calculation works best for data from tests without rest intervals (e.g., ramp tests) and with attached load protocol data.

Value

A `data.frame` with the maximum parameter values of the data.

Examples

```
# Import and process example data sets
gxt_data <- spiro(file = spiro_example("zan_gxt"))

spiro_max(gxt_data)

# Use an averaging over a time interval of 15 seconds
spiro_max(gxt_data, smooth = 15)

# Use an averaging over an interval of 15 breaths
spiro_max(gxt_data, smooth = "15b")
```

---

spiro_plot *Plot data from cardiopulmonary exercise data files*

Description

`spiro_plot()` returns a `ggplot2` graph visualizing data from cardiopulmonary exercise testing.

Usage

```
spiro_plot(
  data,
  which = 1:9,
  smooth = "fz",
  base_size = 13,
  style_args = list(),
  grid_args = list(),
  vert_lines = FALSE
)
```

Arguments

`data`	A `data.frame` of the class `spiro` containing the gas exchange data. Usually the output of a `spiro` call.
`which`	A numeric integer setting the plot panels to be displayed. The panels are numbered in the order of the traditional Wasserman 9-Panel Plot: • 1: VE over time • 2: HR and oxygen pulse over time • 3: VO2, VCO2 and load over time • 4: VE over VCO2 • 5: V-Slope: HR and VCO2 over VO2

- 6: EQVO2 and EQVCO2 over time
- 7: VT over VE
- 8: RER over time
- 9: PetO2 and PetCO2 over time

`smooth` Parameter giving the filter methods for smoothing the data. Default is `fz` for a zero phase Butterworth filter. See `spiro_smooth` for more details and other filter methods (e.g. time based averages)

`base_size` An integer controlling the base size of the plots (in pts).

`style_args` A list of arguments controlling the color and size of lines and points. See the section **'Customization'** for possible arguments. Additional arguments are passed to ggplot2::theme() to modify the appearance of the plots.

`grid_args` A list of arguments passed to `cowplot::plot_grid()` to modify the arrangement of the plots.

`vert_lines` Whether vertical lines should be displayed at the time points of the first warm-up load, first load, and last load. Defaults to FALSE.

Details

This function provides a shortcut for visualizing data from metabolic carts processed by the `spiro` function.

Customization:

There are three ways to customize the appearance of plots in `spiro_plot`. First, you can control the color and size of points and lines with the `style_args` argument. For a list of available arguments that should be passed in form of a list, see below. Second, you can change the appearance of axis and plot elements (e.g, axis titles, panel lines) by passing arguments over to `ggplot2::theme()` via the `style_args` argument. Third, you can modify the arrangement of plots by the `which` argument and customize the arrangement by passing arguments to `cowplot::plot_grid()` via the `grid_args` argument.

Style arguments:

`size = 2` Defines the size of all points

`linewidth = 1` Defines the width of all lines

`color_VO2 = "#c00000", color_VCO2 = "#0053a4", color_VE = "#003300", color_VT = "grey30", color_RER = "#0`
Define the color of lines and points in the following plot panels: VO2 (panel 3,6,9), VCO2 (3,4,5,6,9), VE (1), VT (7), RER (8), HR (2,5), pulse (2)

Additional arguments Are passed to `ggplot2::theme()`

Value

A ggplot object.

Examples

```
# Import and process example data
ramp_data <- spiro(
  file = spiro_example("zan_ramp"),
  hr_file = spiro_example("hr_ramp.tcx")
```

```
)

# Display the traditional Wasserman 9-Panel Plot
spiro_plot(ramp_data)

# Display selected panels, here V-Slope
spiro_plot(ramp_data, which = 5)

# Modify the arrangement of plots by passing arguments to
# cowplot::plot_grid() via the grid_args argument
spiro_plot(ramp_data, which = c(4, 5, 6, 8), grid_args = list(nrow = 1))

# Modify the appearance of plots using the style_args argument
spiro_plot(ramp_data, style_args = list(size = 0.3, color_VCO2 = "black"))

# Modify the appearance of plots by passing arguments to ggplot2::theme() via
# the style_args argument
spiro_plot(ramp_data,
  style_args = list(axis.title.x = ggplot2::element_text(colour = "green"))
)
```

`spiro_raw` *Get raw data from a metabolic cart file or an imported spiro object*

Description

`spiro_raw()` retrieves cardiopulmonary raw data from various types of metabolic cart files, or from objects previously imported and processed with `spiro`.

Usage

```
spiro_raw(data, device = NULL, anonymize = TRUE)

## Default S3 method:
spiro_raw(data, device = NULL, anonymize = TRUE)

## S3 method for class 'spiro'
spiro_raw(data, device = NULL, anonymize = TRUE)
```

Arguments

`data`	Either the absolute or relative path of the file that contains the gas exchange data, or a data frame of the class `spiro`, usually the output of the `spiro` function.
`device`	A character string, specifying the device for measurement. By default the device type is guessed by the characteristics of the `file`. This can be overridden by setting the argument to `"cortex"`, `"cosmed"`, `"vyntus"` or `"zan"`.
`anonymize`	Whether meta data should be anonymized during import. Defaults to TRUE. See `get_anonid` for more information.

Details

The default way of importing data into the spiro package is using the spiro function. Besides importing this will perform further processing steps such as the interpolation of data or the calculation of additional variables. But in some cases the original raw data may be preferable compared to the processed raw data. spiro_raw can either retrieve the raw data from an already imported data set or from a new raw data file.

Value

A data.frame with data. The attribute info contains addition meta-data retrieved from the original file.

Methods (by class)

- spiro_raw(default): Method for direct import from metabolic cart raw data file
- spiro_raw(spiro): Method for objects of class spiro, usually files previously imported and processed with spiro

Examples

```
# Get example data
file <- spiro_example("zan_gxt")

# direct import of raw data
out <- spiro_raw(file)
head(out)

# retrieval of raw data from previously processed object
s <- spiro(file)
out2 <- spiro_raw(s)
head(out2)
```

---

spiro_smooth *Apply a smoothing filter to data from cardiopulmonary exercise testing.*

Description

Filter vectors and data frames with moving averages and digital filters. Provides the data filtering for spiro_max and spiro_plot.

Usage

```
spiro_smooth(data, smooth = 30, columns = NULL, quiet = FALSE)
```

Arguments

data	A data frame of the class spiro. Usually the output of the spiro function.
smooth	An integer or character string specifying the smoothing strategy and parameters. Default is 30, which means the applied filter is a 30-second moving average. See the section **'Filtering Methods'** for more details.
columns	A character vector of the data columns that should be filtered. By default the filtering applies to all data column of data (besides load, time and step).
quiet	Whether warning message should be suppressed. Default is FALSE.

Details

Raw data from cardiopulmonary is usually noisy due to measurement error and biological breath-to-breath variability. When processing or visualizing the gas exchange data, it is often helpful to filter the raw data. This function provides different filtering methods (time average, breath average, digital filters).

Breath-based and digital filters will be applied on the raw breath-by-breath data. Time-based averages will be used on the interpolated data.

Value

A data frame

Filtering Methods

Time-Based Average (e.g. `smooth = 30`**)** A (centered) moving average over a defined time span. The number can be given as an integer or as a character (e.g. `smooth = "30"`) and defines the length of the calculation interval in seconds.

Breath-Based Average (e.g. `smooth = "15b"`**)** A (centered) moving average over a defined number of breaths. The integer before the letter 'b' defines the number of breaths for the calculation interval.

Butterworth filter (e.g. `smooth = "0.04f3"`**)** A digital Butterworth filter (with lag). The number before the letter 'f' defines the low-pass cut-off frequency, the number after the letter 'f' gives the order of the filter. See bw_filter for more details.

Zero-lag Butterworth filter (e.g. `smooth = "0.04fz3"`**)** A digital forwards-backwards Butterworth filter (without lag). The number before the letter 'f' defines the low-pass cut-off frequency, the number after gives the order of the filter. See bw_filter for more details.

Examples

```
# Get example data
file <- spiro_example("zan_gxt")
d <- spiro(file)

out <- spiro_smooth(d, 30)
head(out)

# filter only the VO2 column with a zero-phase Butterworth filter
```

```
out2 <- spiro_smooth(d, "0.04fz3", columns = "VO2")
head(out2)
```

`spiro_summary` *Summarize data from cardiopulmonary exercise testing for each load step*

Description

`spiro_summary()` returns a `data.frame` summarizing the main parameters for each step of a cardiopulmonary exercise test.

Usage

```
spiro_summary(data, interval = 120, quiet = FALSE, exclude = FALSE)
```

Arguments

`data`	A `data.frame` of the class `spiro`, as it is generated by `spiro`.
`interval`	An integer giving the length of the computational interval in seconds.
`quiet`	A logical value, whether or not messages should be displayed, for example when intervals are shortened for specific steps.
`exclude`	A logical value, whether the last step should be excluded from the summary if it was not completely performed.

Details

This function generates mean values of gas exchange and cardiac parameters for all steps of an exercise test. The calculation returns the mean of a given `interval` before the end of each step.

If the interval exceeds the duration of any step, a message will be displayed. If the interval exceeds the duration of all steps, it will be reset to the duration of the longest step. You can silence all messages by setting `quiet = TRUE`.

When setting `exclude = TRUE` the function will check whether the last load step was terminated early. If this was the case, the step will not be displayed in the summary.

Value

A `data.frame` with the mean parameters for each step of the exercise protocol.

Examples

```
# Import and process example data
gxt_data <- spiro(file = spiro_example("zan_gxt"))

spiro_summary(gxt_data)
```

Index