Package: respR (via r-universe)

September 11, 2024

Type Package

Title Import, Process, Analyse, and Calculate Rates from Respirometry Data

Date 2024-05-01

Version 2.3.3

Maintainer Nicholas Carey <`nicholascarey@gmail.com`>

Description Provides a structural, reproducible workflow for the processing and analysis of respirometry data. It contains analytical functions and utilities for working with oxygen time-series to determine respiration or oxygen production rates, and to make it easier to report and share analyses. See Harianto et al. 2019 <doi:10.1111/2041-210X.13162>.

URL `https://github.com/januarharianto/respr`, `https://januarharianto.github.io/respR/`, `https://doi.org/10.1111/2041-210X.13162`

Depends R (>= 3.3)

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE)

Imports data.table, dplyr, glue, graphics, grDevices, lubridate, magrittr, marelac, purrr, roll, segmented, stats, stringr, utils, xml2

Suggests knitr, rmarkdown, rMR, FishResp, respirometry, testthat, covr

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

RemoteUrl https://github.com/januarharianto/respr

RemoteRef HEAD

RemoteSha 14d8c7108a12c42950b5984822fcc812ad3085ba

Contents

`adjust_rate` *Adjust rates to account for background respiration or oxygen flux.*

Description

The `adjust_rate` function adjusts oxygen uptake or production rates (for example, as determined in `calc_rate()` or `auto_rate()`) for background oxygen use by microbial organisms, or for other removal or input of oxygen during a respirometry experiment. The function accepts numeric values, as well as regular `respR` objects, and data frames. See `calc_rate.bg()` for determining background rates, which is the recommended way of passing background rates to `adjust_rate`. Rates determined in `calc_rate` are also accepted as background rates.

Usage

```
adjust_rate(
  x,
  by,
  method = NULL,
  by2 = NULL,
  time_x = NULL,
  time_by = NULL,
  time_by2 = NULL
)
```

Arguments

`x`	numeric. A single numeric value, numeric vector, or object of class `calc_rate`, `calc_rate.int`, `auto_rate`, or `auto_rate.int`. This contains the experimental rate value(s) to be adjusted.
`by`	numeric. A single numeric value, numeric vector, or object of class `calc_rate.bg` or `calc_rate`. This is the background rate(s) used to perform the adjustment to `x`. Can also be a `data.frame` or `inspect` object for `"concurrent"`, `"linear"` or `"exponential"` adjustments. See Details.
`method`	string. Method of background adjustment. Defaults to `"mean"`. Other inputs are: `"value"`, `"paired"`, `"concurrent"`, `"linear"`, `"exponential"`. See Details.
`by2`	numeric. Either a single numeric value, a `calc_rate.bg` or `calc_rate` object, a `data.frame`, or `inspect` object. This is the source of the second background adjustment rate, and used only for dynamic adjustments (`"linear"` or `"exponential"`). See Details.
`time_x`	numeric. The timestamp(s) for the rate(s) in `x`, if it was entered as a numeric (otherwise it is extracted from the `x` input object). Generally this is the midpoint of the time range over which each `x` rate was calculated. Used only in dynamic adjustments (`"linear"` or `"exponential"`). See Details.
`time_by`	numeric. The timestamp of the background correction rate in `by`, if it was entered as a numeric (otherwise it is extracted from the `by` input object). Generally the midpoint of the time range over which it was calculated. Used only in dynamic adjustments (`"linear"` or `"exponential"`). See Details.
`time_by2`	numeric. The timestamp of the background correction rate in `by2`, if it was entered as a numeric (otherwise it is extracted from the `by2` input object). Generally the midpoint of the time range over which it was calculated. Used only in dynamic adjustments (`"linear"` or `"exponential"`). See Details.

Details

`adjust_rate` allows the rate, or multiple rates, in `x` to be adjusted in a number of ways, as detailed below. Note that for those methods which accept them, `by` and `by2` inputs of class `calc_rate`, `calc_rate.bg`, `data.frame` or `inspect` can contain multiple columns of background oxygen data, as long as they share the same numeric time data in column 1. In this case, the mean of all rates calculated for all oxygen columns is used to perform adjustments (see `inspect()` and `calc_rate.bg()` to coerce data to this form). The exception to this is the `"paired"` method, where each rate in `by` (i.e. rate in each oxygen column) is paired with the rate at the same position in `x` and used to adjust it.

Note: take special care with the *sign* of the rate used for adjustments. In `respR` oxygen uptake rates are negative, as they represent a negative slope of oxygen against time. Background rates will normally also be a negative value, while any input of oxygen would be positive. See Examples.

Methods

There are six methods of adjustment, briefly summarised here, with more detail below:

`"value"` - All experimental rates in `x` are adjusted by a single background rate value in `by`.

`"mean"` - This is the default method. All experimental rates in `x` are adjusted by the mean of all background rate values in `by`.

`"paired"` - Experimental rates in `x` are adjusted by the background rate value at the same position in `by`. Therefore requires `x` and `by` to have the same number of rates.

`"concurrent"` - Experimental rates in `x` are adjusted by a background rate calculated over the same time window in the data in `by`. Therefore requires `x` and `by` to share the same time data and length (broadly speaking).

`"linear"` - The time values for experimental rates in `x` are used to calculate an adjustment value based on a background rate that changes *linearly* with respect to time over the course of an experiment. Requires two background recordings or values (`by`, `by2`), and that all data share the same time data or scale.

`"exponential"` - The time values for experimental rates in `x` are used to calculate an adjustment value based on a background rate that changes *exponentially* with respect to time over the course of an experiment. Requires two background recordings or values (`by`, `by2`), and that all data share the same time data or scale.

More Detail

`"value"` - For experiments in which the rate from a single background experiment (or any single background value) is being used to adjust one or more specimen rates. Each rate in `x` is adjusted by the subtracting the single value in `by`. `x` can be a numeric value, numeric vector, `calc_rate`, `calc_rate.int`, `auto_rate`, or `auto_rate.int` object. `by` can be a single numeric value, a `calc_rate.bg` object containing a single `$rate.bg` (i.e. calculated from a 2-column data frame of time~oxygen), or a `calc_rate` object containing a single `$rate`. All other inputs should be `NULL`.

`"mean"` - For experiments in which the mean rate from multiple background experiments is being used to adjust one or more specimen rates. Each rate in `x` is adjusted by subtracting the *mean* of all background rates in `by`. `x` can be a numeric value, numeric vector, `calc_rate`, `calc_rate.int`, `auto_rate`, or `auto_rate.int` object. `by` can be a numeric value, numeric vector, `calc_rate.bg` object containing multiple `$rate.bg`, or a `calc_rate` object containing multiple `$rate`. All other inputs should be `NULL`. If `by` is a single value, this will obviously have the same output as the `"value"` method.

`"paired"` - For experiments where multiple specimen experiments are being adjusted by multiple different background rates. This is a vectorised adjustment operation: rates in `x` are adjusted by the background rates at the same position in `by`. That is, the first `x` adjusted by the first `by`, second `x` by second `by`, etc. `x` can be a numeric value, numeric vector, `calc_rate`, `calc_rate.int`, `auto_rate`, or `auto_rate.int` object. `by` can be a numeric vector *of the same length*, a `calc_rate.bg` or `calc_rate` object where the `$rate.bg` or `$rate` element is the *same length* as the rates in `x` to be adjusted. All other inputs should be `NULL`.

`"concurrent"` - For experiments in which one or more concurrent "blanks" or background experiments are run alongside specimen experiments. Rates in `x` are adjusted by a background rate calculated over the same time window in the data in `by`. That is, the start and end time of each `x` rate is used to fit a linear regression and calculate a background rate in the `$dataframe` in `by`. `x` must be an `calc_rate`, `calc_rate.int`, `auto_rate`, or `auto_rate.int` object. `by` must be a `data.frame`, `inspect`, `calc_rate.bg`, or `calc_rate` object containing time~oxygen data. If there are multiple columns of background oxygen the mean rate across the same time window in all columns is used. In `calc_rate.bg` and `calc_rate` objects the `$rate.bg` or `$rate` element is not used, only the `$dataframe`. The `x` and `by` data must share (broadly) the *same time data or scale in the same units*. If the `x` and `by` data differ in length by more than 5% or some time values are not shared between the two datasets, a warning is given, but the adjustment is nevertheless performed using the available data, by using the closest matching time window in the background data.

`"linear"` - This is a dynamic adjustment, intended for experiments in which the background oxygen rate *changes* over the course of the experiment *linearly* with respect to time. This is typical of long duration respirometry experiments in high temperatures, where a "blank" is conducted at the start of the experiment before the specimen is put in, and again at the end after it is taken out. It requires therefore two background recordings sharing the same numeric *time data* or *time scale*, in the same units as the experiment to be adjusted. These can also be entered as two rate *values* with associated *timestamps*, which again must share the same time scale and units as the rate to be adjusted. This method can also be used in experiments in which a concurrent blank experiment is conducted alongside specimen experiments (as described in the `concurrent` method above), but in which the background data is deemed too noisy to fit reliable regressions over the short timescales specimen rates are determined. In this case, *any* two reliable segments of the background data of any duration can be used to determine how the background rate changes over the course of the experiment, and then this used to adjust specimen rates using the appropriate rate timestamps. The *time~background rate* linear relationship is calculated using the midpoint of the time range of the `by` and `by2` rate regressions (or values plus timestamps). The adjustments to `x` rates are calculated by taking the midpoint of the time range over which it was determined and applying it to the `by~by2` linear relationship. The `x` input can be a numeric value, numeric vector, or a `calc_rate`, `calc_rate.int`, `auto_rate`, or `auto_rate.int` object containing single or multiple rates. The `by` input is the first background recording or rate value, and `by2` the second background recording or rate value.

While it is typical, the `x` rates do not necessarily need to be at intermediate timepoints to the `by`/`by2` times. these are used only to establish a *time~background rate* linear relationship, which can be extrapolated before or after the time values used to calculate it. The `by` and `by2` inputs can be a `data.frame`, `inspect` or `calc_rate.bg` object containing background time~oxygen data. Alternatively, the rate `x`, and background rates `by` and `by2` can be entered as values, in which case the associated timepoints at which these were determined (generally the midpoint of the time range over which the linear regression was fit) must be entered as `time_x`, `time_by`, and `time_by2` (these timepoints are otherwise automatically extracted from the input objects). Multiple `x` rates with multiple `time_x` timepoints can be entered and adjusted, but only one linear background rate relationship applied, that is `by`, `by2`, `time_by`, and `time_by2` must be single numeric values in the correct units.

`"exponential"` - This is a dynamic adjustment, intended for experiments in which the background oxygen rate *changes* over the course of the experiment *exponentially* with respect to time. This is typical of long duration respirometry experiments in high temperatures, where a "blank" is conducted at the start of the experiment before the specimen is put in, and again at the end after it is taken out, and the background rate is found to increase exponentially. This is identical to the `"linear"` method (see above for requirements), except the adjustment is calculated as an exponential relationship of the form - `lm(log(c(by, by2)) ~ c(time_by, time_by2))`.

S3 Generic Functions:

Saved output objects can be used in the generic S3 functions `print()`, `summary()`, and `mean()`.

- `print()`: prints a single result, by default the first adjusted rate. Others can be printed by passing the `pos` input. e.g. `print(x, pos = 2)`
- `summary()`: prints summary table of all results and metadata, or those specified by the `pos` input. e.g. `summary(x, pos = 1:5)`. The summary can be exported as a separate dataframe by passing `export = TRUE`.
- `mean()`: calculates the mean of all adjusted rates, or those specified by the `pos` input. e.g. `mean(x, pos = 1:5)` The mean can be exported as a separate value by passing `export = TRUE`.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at https://januarharianto.github.io/respR/

Value

Output is a list object of class `adjust_rate` containing all inputs, input rates, adjustment values, adjustment method and model (if relevant), and the primary output of interest `$rate.adjusted`.

Examples

```
# Note that oxygen uptake rates are negative in respR since they represent a
# decrease in dissolved oxygen and negative slope. Typically both
# specimen rate and background rate values are negative.

# Simple background adjustment to a single rate
# This is (-7.44) - (-0.04) = -7.40
adjust_rate(x = -7.44, by = -0.04, method = "value")

# Oxygen input adjustment
# This is (-7.44) - (0.1) = -7.54
adjust_rate(x = -7.44, by = 0.1, method = "value")

# Mean background respiration correction to a single rate.
adjust_rate(x = -7.44, by = c(-0.04, -0.05, -0.06),
            method = "mean")
```

```
# Mean background respiration correction to multiple rates.
out <- adjust_rate(x = c(-7.44, -7.20, -7.67),
                   by = c(-0.04, -0.05, -0.06),
                   method = "mean")
summary(out)

# Paired background respiration correction to multiple rates.
out <- adjust_rate(x = c(-7.44, -7.20, -7.67),
                   by = c(-0.04, -0.05, -0.06),
                   method = "paired")
summary(out)

# Dynamic linear adjustment
# With a linear relationship between the 'by' and 'by2' rates,
# at the midpoint time value the adjustment to 'x' should be -0.5
adjust_rate(x = -10,
            time_x = 500,
            by = 0, by2 = -1,
            time_by = 0, time_by2 = 1000,
            method = "linear")

# Same operation to multiple rates
out <- adjust_rate(x = c(-10, -11, -12),
                   time_x = c(500, 600, 700),
                   by = 0, by2 = -1,
                   time_by = 0, time_by2 = 1000,
                   method = "linear")
summary(out)

# A complete workflow using objects instead of values.

# Extract a single replicate from the middle of the zebrafish data
# and calculate rates
zeb_rate <- subset_data(zeb_intermittent.rd,
                        from = 38300,
                        to = 38720,
                        by = "time") %>%
  inspect() %>%
  auto_rate()

# Calculate background rate at start of experiment
bg_start <- subset_data(zeb_intermittent.rd, 1, 4999, "time") %>%
  inspect() %>%
  calc_rate.bg() %>%
  print()

# Calculate background rate at end of experiment
bg_end <- subset_data(zeb_intermittent.rd, 75140, 79251, "time") %>%
  inspect() %>%
  calc_rate.bg() %>%
  print()

# Perform a dynamic linear adjustment
```

```
adjust_rate(zeb_rate, by = bg_start, by2 = bg_end,
            method = "linear") %>%
  summary()

# Note the adjustment values applied are somewhere between the
# start and end background rate values
```

adjust_rate.ft *Adjust rates in flowthrough respirometry to account for background respiration or oxygen flux.*

Description

The `adjust_rate.ft` function adjusts an oxygen uptake or production rate (for example, as determined in `calc_rate.ft()`) for background oxygen use by microbial organisms, or other removal or input of oxygen during *flowthrough* respirometry experiments. The function accepts numeric values, as well as `calc_rate.ft` objects. Numeric `x` and `by` inputs should be rates calculated as the **oxygen delta * flowrate**. Units will be specified in `convert_rate.ft()` when rates are converted to specific output units.

Usage

```
adjust_rate.ft(x, by)
```

Arguments

`x`	numeric. A single numeric value, numeric vector, or object of class `calc_rate.ft`. This is the experimental rate value(s) to be adjusted.
`by`	numeric. A numeric value, numeric vector, or object of class `calc_rate.ft`. This contains the background rate used to perform the adjustment to `x`. If the vector or `calc_rate.ft` object contains multiple rates, they will be averaged to produce a single adjustment value.

Details

`adjust_rate.ft` allows the rate, or multiple rates, in `x` to be adjusted by the background rate in `by`. There are several ways of determining the background rate, or performing background corrections depending on the setup of the experiment.

For experiments in which an empty "blank" experiment has been run, and the background rate generally does not change over the course of the experiment (that is, the oxygen delta between inflow and outflow concentrations remains consistent), it is recommended the rate be determined and saved via the `inspect.ft()` and `calc_rate.ft()` functions and then entered as the `by` input as either a value or the saved `calc_rate.ft` object. In this case, the `$rate` element of the `calc_rate.ft` object is used to adjust all rates in `x`. If there are multiple background rates in `$rate`, the mean value is used. In this way, a single blank experiment can be applied to several specimen experiments. Alternatively, the rate from several blank experiments can be averaged to provide a single adjustment value, and this entered via `by` as a numeric value.

For experiments in which an empty "blank" experiment has been run alongside actual experiments in parallel, and background rate may increase or decrease over time (or there may be other variations for example in the inflow oxygen concentrations), it is recommended you *NOT* use this function. Instead, the paired blank oxygen concentration data should be used in `inspect.ft` as the `in.oxy` input. In this way, the calculated specimen delta oxygen values take account of whatever background or other variation in oxygen is occurring in the blank chamber with respect to time. See examples in the vignettes on the website.

For adjustments, all rates in `x`, whether entered as values or as a `calc_rate.ft` object, are adjusted by subtracting the mean of all background rates in `by`.

Note: take special care with the *sign* of the rate used for adjustments. In `respR` oxygen uptake rates are negative, as they represent a negative slope of oxygen against time. Background rates will normally also be a negative value (though not always). See Examples.

S3 Generic Functions:

Saved output objects can be used in the generic S3 functions `print()`, `summary()`, and `mean()`.

- `print()`: prints a single result, by default the first adjusted rate. Others can be printed by passing the `pos` input. e.g. `print(x, pos = 2)`. See `help("print.adjust_rate.ft")`.
- `summary()`: prints summary table of all results and metadata, or those specified by the `pos` input. e.g. `summary(x, pos = 1:5)`. The summary can be exported as a separate dataframe by passing `export = TRUE`. See `help("summary.adjust_rate.ft")`.
- `mean()`: calculates the mean of all adjusted rates, or those specified by the `pos` input. e.g. `mean(x, pos = 1:5)` The mean can be exported as a separate value by passing `export = TRUE`. See `help("mean.adjust_rate.ft")`.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at `https://januarharianto.github.io/respR/`

Value

Output: If the `x` input is a `calc_rate.ft` object, the output will be identical in structure, but of class `adjust_rate.ft` and containing the additional elements `$adjustment` and `$rate.adjusted`, with these also added to `$summary` metadata.

If `x` is a numeric value or vector, the output is a `list` object of class `adjust_rate.ft` containing four elements: a `$summary` table, `$rate`, `$adjustment`, and `$rate.adjusted`.

For all outputs, the `$rate.adjusted` element will be the one converted when the object is passed to `convert_rate.ft`.

Examples

```
# Note that oxygen uptake rates are negative in respR
# since they represent a decrease in dissolved oxygen
# and negative slope. Typically both specimen rate and
# background rate values are negative.

# ----------------------------------------------------
# Simple background respiration correction to a single
# rate.
# Note, 'x' and 'by' should both be rates calculated as
# the delta oxygen value, the difference between inflow
# and outflow oxygen, multiplied by the flowrate.

# This is (-0.98) - (-0.04) = -0.94
adjust_rate.ft(x = -0.98, by = -0.04)

# ---------------------------------------------------
# Mean background adjustment to a single rate.
adjust_rate.ft(x = -0.98, by = c(-0.04, -0.05, -0.06))

# ---------------------------------------------------
# Mean background adjustment to multiple rates.
out <- adjust_rate.ft(x = c(-0.98, -0.87, -0.91),
                      by = c(-0.04, -0.05, -0.06))
summary(out)

# ---------------------------------------------------
# Adjustment using calc_rate.ft objects
# Specimen rate
sp_rate <- flowthrough_mult.rd %>%
  inspect.ft(time = 1, out.oxy = 2, in.oxy = 6) %>%
  calc_rate.ft(from = 30, flowrate = 0.1)

# Background rate
bg_rate <- flowthrough_mult.rd %>%
  inspect.ft(time = 1, out.oxy = 5, in.oxy = 9) %>%
  calc_rate.ft(flowrate = 0.1)

# Perform adjustment
adj_rate <- adjust_rate.ft(sp_rate, by = bg_rate)
print(adj_rate)
summary(adj_rate)
# ---------------------------------------------------
```

algae.rd *Oxygen production respirometry data*

Description

Data from a respirometry experiment on algae which shows oxygen production over time.

Usage

```
algae.rd
```

Format

A data frame object consisting of 1200 rows (20 h of data),and 2 columns: `$Time` in hours, `$Oxygen` in % air saturation.

Details

- Dissolved oxygen units: `% Air Saturation`
- Time units: `hours`
- Respirometer volume (L): `0.1`
- Temperature (°C): `12`
- Salinity: `30`

Author(s)

Nicholas Carey

`auto_rate` *Automatically determine most linear, highest, lowest and rolling oxygen uptake or production rates*

Description

`auto_rate` performs rolling regressions on a dataset to determine the *most linear, highest, lowest, maximum, minimum, rolling*, and *interval* rates of change in oxygen against time. A rolling regression of the specified `width` is performed on the entire dataset, then based on the "`method`" input, the resulting regressions are ranked or ordered, and the output summarised.

Usage

```
auto_rate(x, method = "linear", width = NULL, by = "row", plot = TRUE, ...)
```

Arguments

`x`	data frame, or object of class `inspect` containing oxygen~time data.
`method`	string. `"linear"`, `"highest"`, `"lowest"`, `"maximum"`, `"minimum"`, `"rolling"` or `"interval"`. Defaults to `"linear"`. See Details.
`width`	numeric. Width of the rolling regression. For `by = "row"`, either a value between 0 and 1 representing a proportion of the data length, or an integer of 2 or greater representing an exact number of rows. If `by = "time"` it represents a time window in the units of the time data. If `NULL`, it defaults to 0.2 or a window of 20% of the data length. See Details.
`by`	string. `"row"` or `"time"`. Defaults to `"row"`. Metric by which to apply the `width` input if it is above 1.
`plot`	logical. Defaults to TRUE. Plot the results.
`...`	Allows additional plotting controls to be passed, such as `pos`, `panel`, and `quiet = TRUE`.

Details

Ranking and ordering algorithms:

Currently, `auto_rate` contains seven ranking and ordering algorithms that can be applied using the `method` input:

- `linear`: Uses kernel density estimation (KDE) to learn the shape of the entire dataset and *automatically identify* the most linear regions of the timeseries. This is achieved by using the smoothing bandwidth of the KDE to re-sample the "peaks" in the KDE to determine linear regions of the data. The summary output will contain only the regressions identified as coming from linear regions of the data, ranked by order of the KDE density analysis. This is present in the `$summary` component of the output as `$density`. Under this method, the `width` input is used as a starting seed value, but the resulting regressions may be of any width. See here for full details.
- `highest`: Every regression of the specified `width` across the entire timeseries is calculated, then ordered using **absolute** rate values from highest to lowest. Essentially, this option ignores the sign of the rate, and can only be used when rates all have the same sign. Rates will be ordered from highest to lowest in the `$summary` table regardless of if they are oxygen uptake or oxygen production rates.
- `lowest`: Every regression of the specified `width` across the entire timeseries is calculated, then ordered using **absolute** rate values from lowest to highest. Essentially, this option ignores the sign of the rate, and can only be used when rates all have the same sign. Rates will be ordered from lowest to highest in the `$summary` table regardless of if they are oxygen uptake or oxygen production rates.
- `maximum`: Every regression of the specified `width` across the entire timeseries is calculated, then ordered using **numerical** rate values from maximum to minimum. Takes **full account of the sign of the rate**. Therefore, oxygen uptake rates, which in `respR` are negative, would be ordered from lowest (least negative), to highest (most negative) in the summary table in numerical order. Therefore, generally this method should only be used when rates are a mix of oxygen consumption and production rates, such as when positive rates may result from regressions fit over flush periods in intermittent-flow respirometry. Generally, for most analyses where maximum or minimum rates are of interest the `"highest"` or `"lowest"` methods should be used.
- `minimum`: Every regression of the specified `width` across the entire timeseries is calculated, then ordered using **numerical** rate values from minimum to maximum. Takes **full account of the sign of the rate**. Therefore, oxygen uptake rates, which in `respR` are negative, would be ordered from highest (most negative) to lowest (least negative) in the summary table in numerical order. Therefore, generally this method should only be used when rates are a mix of oxygen consumption and production rates, such as when positive rates may result from regressions fit over flush periods in intermittent-flow respirometry. Generally, for most analyses where maximum or minimum rates are of interest the `"highest"` or `"lowest"` methods should be used.
- `rolling`: A rolling regression of the specified `width` is performed across the entire timeseries. No reordering of results is performed.
- `interval`: multiple, successive, non-overlapping regressions of the specified `width` are extracted from the rolling regressions, ordered by time.

Further selection and filtering of results:

For further selection or subsetting of `auto_rate` results, see the dedicated `select_rate()` function, which allows subsetting of rates by various criteria, including r-squared, data region, percentiles, and more.

Units:

There are no units involved in `auto_rate`. This is a deliberate decision. The units of oxygen concentration and time will be specified later in `convert_rate()` when rates are converted to specific output units.

The `width` and `by` inputs:

If `by = "time"`, the `width` input represents a time window in the units of the time data in `x`.

If `by = "row"` and `width` is between 0 and 1 it represents a proportion of the total data length, as in the equation `floor(width * number of data rows)`. For example, 0.2 represents a rolling window of 20% of the data width. Otherwise, if entered as an integer of 2 or greater, the `width` represents the number of rows.

For both `by` inputs, if left as `width = NULL` it defaults to 0.2 or a window of 20% of the data length.

In most cases, `by` should be left as the default `"row"`, and the `width` chosen with this in mind, as it is considerably more computationally efficient. Changing to `"time"` causes the function to perform checks for irregular time intervals at every iteration of the rolling regression, which adds to computation time. This is to ensure the specified `width` input is honoured in the time units and rates correctly calculated, even if the data is unevenly spaced or has gaps.

Plot:

A plot is produced (provided `plot = TRUE`) showing the original data timeseries of oxygen against time (bottom blue axis) and row index (top red axis), with the rate result region highlighted. Second panel is a close-up of the rate region with linear model coefficients. Third panel is a rolling rate plot (note the reversed y-axis so that higher oxygen uptake rates are plotted higher), of a rolling rate of the input `width` across the whole dataset. Each rate is plotted against the middle of the time and row range used to calculate it. The dashed line indicates the value of the current rate result plotted in panels 1 and 2. The fourth and fifth panels are summary plots of fit and residuals, and for the `linear` method the sisth panel the results of the kernel density analysis, with the dashed line again indicating the value of the current rate result plotted in panels 1 and 2.

Additional plotting options:

If multiple rates have been calculated, by default the first (`pos = 1`) is plotted. Others can be plotted by changing the `pos` input either in the main function call, or by plotting the output, e.g. `plot(object, pos = 2)`. In addition, each sub-panel can be examined individually by using the `panel` input, e.g. `plot(object, panel = 2)`.

Console output messages can be suppressed using `quiet = TRUE`. If axis labels or other text boxes obscure parts of the plot they can be suppressed using `legend = FALSE`. The rate in the rolling rate plot can be plotted *not* reversed by passing `rate.rev = FALSE`, for instance when examining oxygen production rates so that higher production rates appear higher. If axis labels (particularly y-axis) are difficult to read, `las = 2` can be passed to make axis labels horizontal, and `oma` (outer margins, default `oma = c(0.4, 1, 1.5, 0.4)`), and `mai` (inner margins, default `mai = c(0.3, 0.15, 0.35, 0.15)`) used to adjust plot margins.

S3 Generic Functions:

Saved output objects can be used in the generic S3 functions `print()`, `summary()`, and `mean()`.

- `print()`: prints a single result, by default the first rate. Others can be printed by passing the `pos` input. e.g. `print(x, pos = 2)`
- `summary()`: prints summary table of all results and metadata, or those specified by the `pos` input. e.g. `summary(x, pos = 1:5)`. The summary can be exported as a separate data frame by passing `export = TRUE`.
- `mean()`: calculates the mean of all rates, or those specified by the `pos` input. e.g. `mean(x, pos = 1:5)` The mean can be exported as a separate value by passing `export = TRUE`.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at https://januarharianto.github.io/respR/

Value

Output is a `list` object of class `auto_rate` containing input parameters and data, various summary data, metadata, linear models, and the primary output of interest `$rate`, which can be background adjusted in `adjust_rate` or converted to units in `convert_rate`.

Examples

```
# Most linear section of an entire dataset
inspect(sardine.rd, time = 1, oxygen =2) %>%
  auto_rate()

# What is the lowest oxygen consumption rate over a 10 minute (600s) period?
inspect(sardine.rd, time = 1, oxygen =2) %>%
  auto_rate(method = "lowest", width = 600, by = "time") %>%
  summary()

# What is the highest oxygen consumption rate over a 10 minute (600s) period?
inspect(sardine.rd, time = 1, oxygen =2) %>%
  auto_rate(method = "highest", width = 600, by = "time") %>%
  summary()

# What is the NUMERICAL minimum oxygen consumption rate over a 5 minute (300s)
# period in intermittent-flow respirometry data?
# NOTE: because uptake rates are negative, this would actually be
# the HIGHEST uptake rate.
auto_rate(intermittent.rd, method = "minimum", width = 300, by = "time") %>%
  summary()

# What is the NUMERICAL maximum oxygen consumption rate over a 20 minute
# (1200 rows) period in respirometry data in which oxygen is declining?
# NOTE: because uptake rates are negative, this would actually be
# the LOWEST uptake rate.
sardine.rd %>%
  inspect() %>%
  auto_rate(method = "maximum", width = 1200, by = "row") %>%
  summary()

# Perform a rolling regression of 10 minutes width across the entire dataset.
# Results are not ordered under this method.
sardine.rd %>%
  inspect() %>%
  auto_rate(method = "rolling", width = 600, by = "time") %>%
  summary()
```

auto_rate.int *Run auto_rate on multiple replicates in intermittent-flow respirometry data*

Description

`auto_rate.int` allows you to run the `auto_rate()` function on multiple replicates in intermittent-flow respirometry. A `wait` and `measure` phase can be specified for each replicate, and the `auto_rate` analysis is performed within the `measure` region.

Usage

```
auto_rate.int(
  x,
  starts = NULL,
  wait = NULL,
  measure = NULL,
  by = "row",
  method = "linear",
  width = NULL,
  n = 1,
  plot = TRUE,
  ...
)
```

Arguments

`x`	object of class `inspect` or `data.frame`. This is the timeseries of paired values of oxygen against time containing multiple replicates from which to calculate rates.
`starts`	Numeric. Row locations or times (in the units of the data in `x`) of the start of each replicate. If a single value it indicates a regular interval in rows or time starting from row 1. If a vector, each entry is the start row or time of an individual replicate. Use of rows or time is controlled via `by`.
`wait`	Numeric. A row length or time duration to be applied at the start of each replicate to *exclude* these data from any rate calculations. Can be a single value to apply the same wait phase to each replicate, or a vector of the same length as `starts` of different wait phases for each replicate. Optional.

`measure`	Numeric. A row length or time duration to be applied at the end of the `wait` phase (if used), and used to exclude the flush period. This is the region within which the `auto_rate` analysis is conducted for each replicate. Can be a single value to apply the same measure phase to each replicate, or a vector of the same length as `starts` of different measure phases for each replicate. Default is `NULL` in which case the entire replicate is used (which is rarely what is wanted).
`by`	String. `"row"` or `"time"`. Controls how `starts`, `wait` and `measure` are applied. It also controls how the `width` is applied in the `auto_rate` analysis - see `help("auto_rate")`. Default is `"row"`.
`method`	string. The `auto_rate` `method` to use. Default is `"linear"`. Others include `"lowest"` and `"highest"`. See `help("auto_rate")` for descriptions and other methods.
`width`	numeric. The `width` to use in the `auto_rate` analysis. Mandatory and should be entered in the correct units of the `by` input. See `help("auto_rate")` and vignettes on website for how width affects analyses.
`n`	integer. How many `auto_rate` results to return for each replicate. Default is `1`.
`plot`	logical. Default is `TRUE`. Plots the results. See 'Plotting' section for details.
`...`	Allows additional plotting controls to be passed, such as `type`, `pos`, `legend`, and `quiet`.

Details

`auto_rate.int` uses the `starts` input to subset each replicate. The `wait` and `measure` inputs control which parts of each replicate data are excluded and included from the rate calculation. It runs `auto_rate` on the `measure` phase in each replicate saving the top `n` ranked results and extracting the rate and other data to a summary table.

The `x` input should be an`inspect` object. Alternatively, it can be a two-column data frame containing paired values of time and oxygen from an intermittent-flow experiment in columns 1 and 2 respectively (though we always recommend processing such data in `inspect()` first). If a multiple column dataset is entered as `x` the first two columns are selected by default. If these are not the intended data use `inspect` to select the correct time and oxygen columns.

`auto_rate` **inputs:**

You should be familiar with how `auto_rate` works before using this function. See `help("auto_rate")` and vignettes on the website for full details.

The `auto_rate` inputs can be changed by entering different `method` and `width` inputs. The `by` input controls how the `width` is applied. Note if using a proportional `width` input (i.e. between 0 and 1 representing a proportion of the data length) this applies to the length of the `measure` phase of each particular replicate.

The `n` input controls how many `auto_rate` results from each replicate to return in the output. By default this is only the top ranked result for the particular `method`, i.e. `n = 1`. This can be changed to return more, however consider carefully if this is necessary as the output will necessarily contain many more rate results which may make it difficult to explore and select results (although see `select_rate()`).

Specifying replicate structure:

The `starts` input specifies the locations of the start of each replicate in the data in `x`. This can be in one of two ways:

- A single numeric value specifying the number of rows in each replicate starting from the data in the first row. This option should only be used when replicates cycle at regular intervals. This can be a regular row or time interval, as specified via the `by` input. If the first replicate does not start at row 1, the data should be subset so that it does (see `subset_data()`) and example here. For example, `starts = 600, by = "row"` means the first replicate starts at row 1 and ends at row 600, the second starts at row 601 ends at 1200, and so on.
- A numeric vector of row locations or times, as specified via the `by` input, of the start of each individual replicate. The first replicate does not have to start at the first row of the data, and all data after the last entry is assumed to be part of the final replicate. Regular `R` syntax such as `seq()`, `1:10`, etc. is also accepted, so can be used to specify both regular and irregular replicate spacing.

For both methods it is assumed each replicate ends at the row preceding the start of the next replicate, or in the case of the last replicate the final row of the dataset. Also for both methods, `by = "time"` inputs do not need to be exact; the closest matching values in the time data are used.

Results are presented in the `summary` table with `rep` and `rank` columns to distinguish those from different replicates and their ranking within replicates (if multiple results per replicate have been returned by increasing the `n` input).

Specifying rate region:

The `wait` and `measure` inputs are used to specify the region from which to extract a rate and exclude flush periods. They can be entered as row intervals or time values in the units of the input data. The `wait` phase controls the amount of data at the start of each replicate to be ignored, that is excluded from any rate calculations. The `measure` phase determines the region after this from which a rate is calculated. Unlike `calc_rate.int()`, `auto_rate.int` will not necessarily use all of the data in the `measure` phase, but will run the `auto_rate` analysis *within* it using the `method`, `width` and `by` inputs. This may result in rates of various widths depending on the inputs. See `auto_rate()` for defaults and full details of how selection inputs are applied.

There is no `flush` phase input since this is assumed to be from the end of the `measure` phase to the end of the replicate.

Both `wait` and `measure` can be entered in one of two ways:

- Single numeric values specifying a row width or a time period, as specified via the `by` input. Use this if you want to use the *same* `wait` and `measure` phases in every replicate.
- If `starts` is a vector of locations of the start of each replicate, these inputs can also be vectors of equal length of row lengths or time periods as specified via the `by` input. This is only useful if you want to use *different* `wait` and/or `measure` phases in different replicates.

If `wait = NULL` no wait phase is applied. If `measure = NULL` the data used for analysis is from the start of the replicate or end of the `wait` phase to the last row of the replicate. This will typically include the flush period, so is rarely what you would want.

Example:

See examples below for actual code, but here is a simple example. An experiment comprises replicates which cycle at ten minute intervals with data recorded every second. Therefore each replicate will be 600 rows long. Flushes of the respirometer take 3 minutes at the end of each replicate. We want to exclude the first 2 minutes (120 rows) of data in each, and run an `auto_rate` analysis to get an oxygen uptake rate within the following five minute period (300 rows), leaving the three minutes of flushing (180 rows) excluded. The inputs for this would be:
`starts = 600, wait = 120, measure = 300, by = "row"`

Plot:

If `plot = TRUE` (the default), the result for each rate is plotted on a grid up to a maximum of 20. There are three ways of plotting the results, which can be selected using the `type` input:

- `type = "rep"`: The default. Each individual replicate is plotted with the rate region highlighted in yellow. The `wait` and `measure` phases are also highlighted as shaded red and green regions respectively. These are also labelled if `legend = TRUE`.
- `type = "full"`: Each replicate rate is highlighted in the context of the whole dataset. May be quite difficult to interpret if dataset is large.
- `type = "ar"`: Plots individual replicate results as `auto_rate` objects. Note, these will only show the `measure` phase of the data.

For all plot types `pos` can be used to select which rate(s) to plot (default is 1:20), where `pos` indicates rows of the `$summary` table (and hence which `$rep` and `$rank`). This can be passed either in the main function call or when calling `plot()` on output objects. Note for all plot types if `n` has been changed to return more than one rate per replicate these will also be plotted.

S3 Generic Functions:

Saved output objects can be used in the generic S3 functions `plot()`, `print()`, `summary()`, and `mean()`. For all of these `pos` selects rows of the `$summary` table.

- `plot()`: plots the result. See Plot section above.
- `print()`: prints the result of a single rate, by default the first. Others can be printed by passing the `pos` input. e.g. `print(x, pos = 2)`
- `summary()`: prints summary table of all results and metadata, or the rows specified by the `pos` input. e.g. `summary(x, pos = 1:5)`. The `$rep` column indicates the replicate number, and `$rank` column the ranking of each rate *within* each replicate (only used if a different `n` has been passed, otherwise they are all `1`). The summary table (or `pos` rows) can be exported as a separate data frame by passing `export = TRUE`.
- `mean()`: calculates the mean of the rates from every row or those specified by the `pos` input. e.g. `mean(x, pos = 1:5)` Note if a different `n` has been passed this may include multiple rates from each replicate. The mean can be exported as a numeric value by passing `export = TRUE`.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at `https://januarharianto.github.io/respR/`

Value

Output is a `list` object of class `auto_rate.int` containing a `auto_rate` object for each replicate in `$results`. The output also contains a `$summary` table which includes the full rate regression results from each replicate with replicate number indicated by the `$rep` column. Output also contains a `$rate` element which contains the rate values from each replicate in order. The function call, inputs, and other metadata are also included. Note, that if you have many replicates this object can be rather large (several MB).

Examples

```
# Irregular replicate structure ------------------------------------------
# Prepare the data to use in examples
# Note in this dataset each replicate is a different length!
data <- intermittent.rd
# Convert time to minutes (to show different options below)
data[[1]] <- round(data[[1]]/60, 2)
# Inspect
urch_insp <- inspect(data)

# Calculate the most linear rate within each replicate
auto_rate.int(urch_insp,
              starts = c(1, 2101, 3901),
              by = "row",
              method = "linear",
              width = 400) %>%
  summary()

# Calculate the lowest rate within each replicate across
# 5 minutes (300 rows). For this we need to specify a 'measure' phase
# so that the flush is excluded.
auto_rate.int(urch_insp,
              starts = c(1, 2101, 3901),
              measure = 1000,
              by = "row",
              method = "lowest",
              width = 300) %>%
  summary()

# You can even specify different 'measure' phases in each rep
auto_rate.int(urch_insp,
              starts = c(1, 2101, 3901),
              measure = c(1000, 800, 600),
              by = "row",
              method = "lowest",
              width = 300) %>%
  summary()

# We usually don't want to use the start of a replicate just after the flush,
# so we can specify a 'wait' phase. We can also specify 'starts', 'wait',
# 'measure', and 'width' in units of time instead of rows.
#
# By time
# (this time we save the result)
urch_res <- auto_rate.int(urch_insp,
                          starts = c(0, 35, 65), # start locations in minutes
                          wait = 2,              # wait for 2 mins
                          measure = 10,          # measure phase of 10 mins
                          by = "time",           # apply inputs by time values
                          method = "lowest",     # get the 'lowest' rate...
                          width = 5) %>%          #  ... of 5 minutes width
  summary()

# Regular replicate structure --------------------------------------------
# If replicates cycle at regular intervals, 'starts' can be used to specify
# the spacing in rows or time, starting at row 1. Therefore data must be
# subset first so that the first replicate starts at row 1.
#
# Subset and inspect data
zeb_insp <- zeb_intermittent.rd %>%
  subset_data(from = 5840,
              to = 75139,
              by = "row",
              quiet = TRUE) %>%
  inspect()

# Calculate the most linear rate from the same 6-minute region in every
# replicate. Replicates cycle at every 660 rows.
zeb_res <- auto_rate.int(zeb_insp,
                         starts = 660,
                         wait = 120, # exclude first 2 mins
                         measure = 360, # measure period of 6 mins after 'wait'
                         method = "linear",
                         width = 200, # starting value for linear analysis
                         plot = TRUE) %>%
  summary()

# S3 functions -------------------------------------------------------------

# Outputs can be used in print(), summary(), and mean().
# 'pos' can be used to select replicate ranges
summary(zeb_res)
mean(zeb_res, pos = 1:5)

# There are three ways by which the results can be plotted.
# 'pos' can be used to select replicates to be plotted.
#
# type = "rep" - the default. Each replicate plotted on a grid with rate
# region highlighted (up to a maximum of 20).
plot(urch_res)

# type = "full" - each replicate rate region plotted on entire data series.
plot(urch_res, pos = 1:2, type = "full")
# Of limited utility when datset is large
plot(zeb_res, pos = 10, type = "full")

# type = "ar" - the 'auto_rate' object for selected replicates in 'pos' is plotted
# Note this shows the 'measure' phase only
plot(urch_res, pos = 2, type = "ar")

# See vignettes on website for how to adjust and convert rates from auto_rate.int
```

background_con.rd *Background respirometry data (constant)*

Description

Background oxygen consumption data. After the initial 30 minutes, data shows a generally constant background rate. Taken from a Loligo swim tunnel background recording. Oxygen recorded via a Witrox sensor in % air saturation over nearly 6 hours at 1 second intervals. Data is from a real experiment.

Usage

```
background_con.rd
```

Format

A data frame object consisting of 20664 rows (approx 6 h of data),and 2 columns: `$Time` in seconds, `$Oxygen` in % air saturation.

Details

- Dissolved oxygen units: `% Air Saturation`
- Time units: `seconds`
- Swim tunnel volume (L): `12.3`
- Temperature (°C): `14.5`
- Salinity: `34`
- Atm. Pressure (bar): `1.013253`

Author(s)

Nicholas Carey

`background_exp.rd` *Background respirometry data (exponential)*

Description

Background oxygen consumption data. Data shows a background rate which increases exponentially with respect to time. Taken from a Loligo swim tunnel background recording. Oxygen recorded via a Witrox sensor in % air saturation over nearly 6 hours at 1 second intervals. Data is from a real experiment, but oxygen decrease curve has been exaggerated to impose an exponential increase in background consumption for testing purposes.

Usage

```
background_exp.rd
```

Format

A data frame object consisting of 20664 rows (approx 6 h of data),and 2 columns: `$Time` in seconds, `$Oxygen` in % air saturation.

Details

- Dissolved oxygen units: `% Air Saturation`
- Time units: `seconds`
- Swim tunnel volume (L): `12.3`
- Temperature (°C): `14.5`
- Salinity: `34`
- Atm. Pressure (bar): `1.013253`

Author(s)

Nicholas Carey

`background_lin.rd` *Background respirometry data (linear)*

Description

Background oxygen consumption data. After initial 30 minutes, data shows a background rate which increases linearly with respect to time. Taken from a Loligo swim tunnel background recording. Oxygen recorded via a Witrox sensor in % air saturation over nearly 6 hours at 1 second intervals. Data is from a real experiment, but has been manipulated to show a linear increase in background rate for testing purposes.

Usage

```
background_lin.rd
```

Format

A data frame object consisting of 20664 rows (approx 6 h of data),and 2 columns: `$Time` in seconds, `$Oxygen` in % air saturation.

Details

- Dissolved oxygen units: `% Air Saturation`
- Time units: `seconds`
- Swim tunnel volume (L): `12.3`
- Temperature (°C): `14.5`
- Salinity: `34`
- Atm. Pressure (bar): `1.013253`

Author(s)

Nicholas Carey

calc_rate *Calculate rate of change in oxygen over time*

Description

Calculates rate of oxygen uptake or production from respirometry data. A rate can be determined over the whole dataset, or on subsets of the data using the `from` and `to` inputs to specify data regions in terms of `oxygen` or `time` units or `row` numbers of the input data. Multiple rates can be extracted from the same dataset by using these inputs to enter vectors of paired values in the appropriate metric. See Examples.

Usage

```
calc_rate(x, from = NULL, to = NULL, by = "time", plot = TRUE, ...)
```

Arguments

`x`	object of class `inspect` or `data.frame`. This is the timeseries of paired values of oxygen against time from which to calculate rates.
`from`	numeric value or vector. Defaults to `NULL`. The start of the region(s) over which you want to calculate the rate in the units specified in `by`. If a vector, each value must have a paired value in `to`.
`to`	numeric value or vector. Defaults to `NULL`. The end of the region(s) over which you want to calculate the rate in the units specified in `by`. If a vector, each value must have a paired value in `from`.
`by`	string. `"time"`, `"row"`, or `"oxygen"`. Defaults to `"time"`. This is the method used to subset the data region between `from` and `to`.
`plot`	logical. Defaults to `TRUE`. Plot the results.
`...`	Allows additional plotting controls to be passed, such as `pos`, `panel`, and `quiet = TRUE`.

Details

The function calculates rates by fitting a linear model of oxygen against time, with the slope of this regression being the rate. There are no units involved in `calc_rate`. This is a deliberate decision. The units of oxygen concentration and time will be specified later in `convert_rate()` when rates are converted to specific output units.

For continuous data recordings, it is recommended a `data.frame` containing the data be prepared via `inspect()`, and entered as the `x` input. For data not prepared like this, `x` can be a 2-column `data.frame` containing numeric values of time (col 1) and oxygen (col 2). If multiple columns are found in either an `inspect` or data frame input, only the first two columns are used.

Specifying regions:

For calculating rates over specific regions of the data, the `from` and `to` inputs in the `by` units of `"time"` (the default), `"oxygen"`, or `"row"`. The `from` and `to` inputs do not need to be precise; the function will use the closest values found.

Multiple regions can be examined within the same dataset by entering `from` and `to` as vectors of paired values to specify different regions. In this case, `$rate` in the output will be a vector of multiple rates with each result corresponding to the position of the paired `from` and `to` inputs. If `from` and `to` are `NULL` (the default), the rate is determined over the entire dataset.

Plot:

A plot is produced (provided `plot = TRUE`) showing the original data timeseries of oxygen against time (bottom blue axis) and row index (top red axis), with the region specified via the `from` and `to` inputs highlighted. Second panel is a close-up of the rate region with linear model coefficients. Third and fourth panels are summary plots of fit and residuals.

Additional plotting options:

If multiple rates have been calculated, by default the first (`pos = 1`) is plotted. Others can be plotted by changing the `pos` input either in the main function call, or by plotting the output, e.g. `plot(object, pos = 2)`. In addition, each sub-panel can be examined individually by using the `panel` input, e.g. `plot(object, panel = 2)`.

Console output messages can be suppressed using `quiet = TRUE`. If axis labels (particularly y-axis) are difficult to read, `las = 2` can be passed to make axis labels horizontal, and `oma` (outer margins, default `oma = c(0.4, 1, 1.5, 0.4)`), and `mai` (inner margins, default `mai = c(0.3, 0.15, 0.35, 0.15)`) used to adjust plot margins.

S3 Generic Functions:

Saved output objects can be used in the generic S3 functions `print()`, `summary()`, and `mean()`.

- `print()`: prints a single result, by default the first rate. Others can be printed by passing the `pos` input. e.g. `print(x, pos = 2)`
- `summary()`: prints summary table of all results and metadata, or those specified by the `pos` input. e.g. `summary(x, pos = 1:5)`. The summary can be exported as a separate dataframe by passing `export = TRUE`.
- `mean()`: calculates the mean of all rates, or those specified by the `pos` input. e.g. `mean(x, pos = 1:5)` The mean can be exported as a separate value by passing `export = TRUE`.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at `https://januarharianto.github.io/respR/`

Value

Output is a `list` object of class `calc_rate` containing input parameters and data, various summary data, metadata, linear models, and the primary output of interest `$rate`, which can be background adjusted in `adjust_rate` or converted to units in `convert_rate`.

Examples

```
# Subset by 'time' (the default)
inspect(sardine.rd, time = 1, oxygen = 2, plot = FALSE) %>%
  calc_rate(from = 200, to = 1800)

# Subset by oxygen
inspect(sardine.rd, time = 1, oxygen = 2, plot = FALSE) %>%
  calc_rate(94, 91, by = "oxygen")

# Subset by row
inspect(sardine.rd, time = 1, oxygen = 2, plot = FALSE) %>%
  calc_rate(1000, 2000, by = "row")

# Use a data frame input, and calculate rate from multiple regions by
# using a vector in the 'from' and 'to' inputs
x <- calc_rate(intermittent.rd,
               from = c(200,2300,4100),
               to = c(1800,3200,4600),
               by = 'time',
               plot = FALSE)
# Print and summary of results
print(x)
summary(x)
# Plot the third of these results
plot(x, pos = 3)
# Plot only the timeseries plot and hide the legend
plot(x, pos = 3, panel = 1, legend = FALSE)
```

calc_rate.bg — *Calculate background oxygen uptake or input rates*

Description

This function calculates the rate of change of oxygen over time from "blank" or control respirometry experiments, to allow for background adjustments of experimental data. It accepts background oxygen~time data as data frames and `inspect` objects. The data must be in the same time and oxygen units as the data from which the rate which will be adjusted was extracted. Multiple columns of background oxygen measurements can be entered as long as they share the same time data. In this case the function returns rates for all columns, and also calculates a mean rate.

Usage

```
calc_rate.bg(x, time = NULL, oxygen = NULL, plot = TRUE, ...)
```

Arguments

`x`	`data.frame` or `inspect` object. This is the data to extract background rate(s) from.
`time`	integer. Defaults to 1. This specifies the column number of the time data.
`oxygen`	integer value or vector. This specifies the column number(s) of the oxygen data. Multiple columns of oxygen can be specified. If NULL, function assumes oxygen data are in *all* columns of the data frame except the `time` column.
`plot`	logical. Defaults to TRUE. Plots the data. See Details.
`...`	Allows additional plotting controls to be passed, such as `pos`, `legend = FALSE`, and `quiet = TRUE`.

Details

The main difference between `calc_rate.bg` and `calc_rate`, is that this function allows a rate to be determined from the same region of multiple oxygen data columns, whereas `calc_rate` allows multiple rates to be determined from different regions of a single dataset.

Units:

There are no units involved in `calc_rate.bg`. This is a deliberate decision. The units of oxygen concentration and time will be specified later in `convert_rate()` when rates are converted to specific output units. It is important however, the background time~oxygen data is in the same time and oxygen units as the data used to determine the rate which will be adjusted.

Subsetting data regions:

`calc_rate.bg` does not have internal subsetting of data regions. If you need to subset the data to specific regions you don't want to use, see `subset_data()`, which allows for easy passing (or piping) of subsets to `calc_rate.bg`.

Background respiration vs background input of oxygen:

Most users will be using this function to account for background oxygen consumption rates from microbial activity that need to be quantified and their effects removed from experimental specimen rates. However, there are some experiments where oxygen *input* rates may be of interest, for example in open tank or open arena respirometry where the input of oxygen from the water surface has been calculated or quantified. There are also cases in closed respirometry where there may be an input of oxygen via leaks or oxygen production from photosynthesis which need to be quantified. `calc_rate.bg` is readily capable of quantifying production rates as well as consumption, and these can also be used for adjustments in `adjust_rate()`.

Plot:

A plot is produced (provided `plot = TRUE`) showing all examined columns of oxygen against time (bottom blue axis) and row index (top red axis), with the rate and linear model coefficients. Single rates can be plotted by changing the `pos` input either in the main function call, or by plotting the output, e.g. `plot(object, pos = 2)`. Console output messages can be suppressed using `quiet = TRUE`. If equations obscure the plot they can be suppressed using `legend = FALSE`.

S3 Generic Functions:

Saved output objects can be used in the generic S3 functions `print()`, `summary()`, and `mean()`.

- `print()`: prints all background rates, plus the mean background rate.
- `summary()`: prints summary table of all results and metadata, or those specified by the `pos` input. e.g. `summary(x, pos = 1:5)`. The summary can be exported as a separate dataframe by passing `export = TRUE`.
- `mean()`: calculates the mean of all rates, or those specified by the `pos` input. e.g. `mean(x, pos = 1:5)` The mean can be exported as a separate value by passing `export = TRUE`.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at https://januarharianto.github.io/respR/

Value

Output is a `list` object of class `calc_rate.bg` containing original data, linear models, summary information, and the primary output of interest `$rate.bg`, which contains a rate for each oxygen column present in the input data. There is also `$rate.bg.mean` containing the mean of all background rates. Note, this is not used in `adjust_rate`, where the `method` input there determines how `$rate.bg` is applied, but can easily be extracted and applied as an adjustment value if desired.

Examples

```
# Inspect and calculate background rate from two columns
inspect(urchins.rd, time = 1, oxygen = 18:19) %>%
  calc_rate.bg()

# Same example but enter as a data frame, save as an object and use
# in adjust_rate
bg_rate <- calc_rate.bg(urchins.rd,
                        time = 1,
                        oxygen = 18:19,
                        plot = FALSE)

inspect(urchins.rd, 1, 2, plot = FALSE) %>%
  calc_rate(from = 10, to = 30, by = "time", plot = FALSE) %>%
  adjust_rate(by = bg_rate)

# Subset single column data first before calculating background rate
subset_data(background_con.rd, from = 5000, to = 20000, by = "time") %>%
  calc_rate.bg()
```

`calc_rate.ft` *Calculate rate of change in oxygen from flowthrough respirometry data*

Description

Calculates rate of oxygen uptake or production in flowthrough respirometry data given a `flowrate` and delta oxygen values, which can either be directly entered, or be calculated from inflow and outflow oxygen. The function returns a single rate value from the whole dataset or a subset of it, by averaging delta oxygen values. Alternatively, multiple rate values can be returned from different regions of continuous data, or a rolling rate of a specific window size performed across the whole dataset.

Usage

```
calc_rate.ft(
  x = NULL,
  flowrate = NULL,
  from = NULL,
  to = NULL,
  by = NULL,
  width = NULL,
  plot = TRUE,
  ...
)
```

Arguments

`x`	numeric value or vector of delta oxygen values, a 2-column `data.frame` of outflow (col 1) and inflow (col 2) oxygen values, or an object of class `inspect.ft`.
`flowrate`	numeric value. The flow rate through the respirometer in volume (ul,ml,L) per unit time (s,m,h,d). The units are not necessary here, but will be specified in `convert_rate.ft`.
`from`	numeric value or vector. Defaults to `NULL`. The start of the region(s) over which you want to calculate the rate in either time or row units. If a vector, each value must have a paired value in `to`. For use with `inspect.ft` inputs only.
`to`	numeric value or vector. Defaults to `NULL`. The end of the region(s) over which you want to calculate the rate in either time or row units. If a vector, each value must have a paired value in `from`. For use with `inspect.ft` inputs only.
`by`	`"time"` or `"row"`. Defaults to `"time"`. Specifies the units of the `from` and `by`, or `width` value. For use with `inspect.ft` inputs only.
`width`	numeric. Calculates a rolling rate across the whole dataset of the specified width in the units specified in `by`. For use with `inspect.ft` inputs only.
`plot`	logical. Defaults to TRUE. Plots the data.
`...`	Allows additional plotting controls to be passed such as `pos`, `quiet = TRUE`, `legend = FALSE`, and `rate.rev = FALSE`.

Details

`calc_rate.ft` calculates rates by averaging delta oxygen values across the whole dataset, or from specified subsets of the data. The `flowrate` is then used to convert these average delta values to rates. There are no units involved in `calc_rate.ft`. This is a deliberate decision. The units of oxygen concentration and flowrate will be specified later in `convert_rate.ft()` when rates are converted to specific output units.

For continuous data recordings, it is recommended a `data.frame` containing the data be prepared via `inspect.ft()`, and entered as the `x` input.

For data not prepared like this, `x` can be a 2-column `data.frame` containing numeric values of outflow (col 1) and inflow (col 2) oxygen concentrations in that order. Alternatively, if `x` is a numeric value or vector it is treated as delta oxygen values (outflow oxygen concentration minus inflow oxygen concentration in the same units). In both these cases, the `from`, `to`, and `by` inputs are are ignored, and all delta oxygen values whether as entered or calculated from the inflow and outflow oxygen columns are converted to rates.

Specifying regions:

For calculating rates over specific regions of the data, the `from` and `to` inputs in the `by` units of `"time"` (the default) or `"row"` can be used for `inspect.ft()` inputs. All delta oxygen values within this region are converted to rates, and averaged to produce a overall rate for the region (`$rate` in the output). Multiple regions can be examined within the same dataset by entering `from` and `to` as vectors of paired values to specify different regions. In this case, `$rate` in the output will be a vector of multiple rates with each result corresponding to the position of the paired `from` and `to` inputs. If `from` and `to` are `NULL` (the default), the rate is determined over the entire dataset.

Alternatively a `width` input can be specified, in which case a rolling rate is calculated using this window size (in the relevant `by` units) across the entire dataset, and returned as a vector of rate values in `$rate`. See here for how this might be used.

Flowrate:

In order to convert delta oxygen values to a oxygen uptake or production rate, the `flowrate` input is required. This must be in a volume (L, ml, or ul) per unit time (s,m,h,d), for example in `L/s`. The units are not required to be entered here; they will be specified in `[convert_rate.ft()]` to convert rates to specific units of oxygen uptake or production.

Plot:

For rates calculated from `inspect.ft` inputs, a plot is produced (provided `plot = TRUE`) showing the original data timeseries of inflow and outflow oxygen (if present, top plot), oxygen delta values (middle or top plot) with the region specified via the `from` and `to` inputs highlighted in orange, and a close-up of this region with calculated rate value (bottom plot). If multiple rates have been calculated, by default the first is plotted. Others can be plotted by changing the `pos` input, e.g. `plot(object, pos = 2)`.

Important: Since `respR` is primarily used to examine oxygen consumption, the delta oxygen and rate plots are by default plotted on a reverse y-axis. In `respR` oxygen uptake rates are negative since they represent a negative slope of oxygen against time. In these plots the axis is reversed so that higher uptake rates (i.e. more negative rates) will be higher on these plots. If you are interested instead in oxygen production rates, which are positive, the `rate.rev = FALSE` input can be passed in either the `inspect.ft` call, or when using `plot()` on the output object. In this case, the delta and rate values will be plotted numerically, with higher oxygen *production* rates higher on the plot.

Additional plotting options:

If the legend or labels obscure part of the plot, they can be suppressed via `legend = FALSE` in either the `inspect.ft` call, or when using `plot()` on the output object. Console output messages can be suppressed using `quiet = TRUE`. Console output messages can be suppressed using `quiet = TRUE`. If axis labels or other text boxes obscure parts of the plot they can be suppressed using `legend = FALSE`. If axis labels (particularly y-axis) are difficult to read, `las = 2` can be passed to make axis labels horizontal, and`oma` (outer margins, default `oma = c(0.4, 1, 1.5, 0.4)`), and `mai` (inner margins, default `mai = c(0.3, 0.15, 0.35, 0.15)`) used to adjust plot margins.

Background control or "blank" experiments:

`calc_rate.ft` can also be used to determine background rates from empty control experiments in the same way specimen rates are determined. The saved objects can be used as the `by` input in `adjust_rate.ft()`. For experiments in which the specimen data is to be corrected by a concurrently-run control experiment, best option is to use this as the `in.oxy` input in `inspect.ft()`. See help file for that function, or the vignettes on the website for examples.

S3 Generic Functions:

Saved output objects can be used in the generic S3 functions `print()`, `summary()`, and `mean()`.

- `print()`: prints a single result, by default the first rate. Others can be printed by passing the `pos` input. e.g. `print(x, pos = 2)`
- `summary()`: prints summary table of all results and metadata, or those specified by the `pos` input. e.g. `summary(x, pos = 1:5)`. The summary can be exported as a separate data frame by passing `export = TRUE`.
- `mean()`: calculates the mean of all rates, or those specified by the `pos` input. e.g. `mean(x, pos = 1:5)` The mean can be exported as a separate value by passing `export = TRUE`.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at `https://januarharianto.github.io/respR/`

Value

Output is a `list` object of class `calc_rate.ft` containing input parameters and data, various summary data, metadata, and the primary output of interest `$rate`, which can be background adjusted in `adjust_rate.ft` or converted to units in `convert_rate.ft`. Note the `$summary` table contains linear regression coefficients alongside other metadata. These *should not be confused* with those in other functions such as `calc_rate` where slopes represent rates and coefficients such as a high r-squared are important. Here, they represent the stability of the data region, in that the closer the slope is to zero the less the delta oxygen values, and therefore rates, in that region vary. These are included to enable possible future functionality where stable regions may be automatically identified.

Examples

```
# Single numeric delta oxygen value. The delta oxygen is the difference
# between inflow and outflow oxygen.
calc_rate.ft(-0.8, flowrate = 1.6)

# Numeric vector of multiple delta oxygen values
ft_rates <- calc_rate.ft(c(-0.8, -0.88, -0.9, -0.76), flowrate = 1.6)
print(ft_rates)
summary(ft_rates)

# Calculate rate from entire dataset
inspect.ft(flowthrough.rd, time = 1, out.oxy = 2, in.oxy = 3, ) %>%
  calc_rate.ft(flowrate = 2.34)

# Calculate rate from a region based on time
inspect.ft(flowthrough.rd, time = 1, out.oxy = 2, in.oxy = 3, ) %>%
  calc_rate.ft(flowrate = 2.34, from = 200, to = 400, by = "time")

# Calculate rate from multiple regions
inspect.ft(flowthrough.rd, time = 1, out.oxy = 2, in.oxy = 3, ) %>%
  calc_rate.ft(flowrate = 2.34,
               from = c(200, 400, 600),
               to = c(300, 500, 700),
               by = "row") %>%
  summary()

# Calculate rate from existing delta oxygen values
inspect.ft(flowthrough.rd, time = 1, delta.oxy = 4) %>%
  calc_rate.ft(flowrate = 2.34, from = 200, to = 400, by = "time")

# Calculate rate from a background recording
inspect.ft(flowthrough_mult.rd,
           time = 1,
           out.oxy = 5,
           in.oxy = 9) %>%
  calc_rate.ft(flowrate = 0.1, from = 20, to = 40, by = "time") %>%
  summary()

# Calculate a rolling rate
inspect.ft(flowthrough_mult.rd,
           time = 1,
           out.oxy = 2,
           in.oxy = 6) %>%
  calc_rate.ft(flowrate = 0.1, width = 500, by = "row") %>%
  summary()
```

---

calc_rate.int *Extract rates from multiple replicates in intermittent-flow respirometry data*

Description

`calc_rate.int` allows you to extract an oxygen uptake or production rate from multiple replicates in intermittent-flow respirometry. It allows you to easily use consistent selection parameters to extract a single rate from each replicate, for example a specific time range or row range.

Usage

```
calc_rate.int(
  x,
  starts = NULL,
  wait = NULL,
  measure = NULL,
  by = "row",
  plot = TRUE,
  ...
)
```

Arguments

x	Object of class `inspect` or `data.frame`. This is the timeseries of paired values of oxygen against time containing multiple replicates from which to calculate rates.

`starts`	Integer(s). Row locations or times of the start of each replicate. A single value input indicates a regular interval in rows or time units starting at the first row of the data in `x`. If replicates do not cycle at a regular interval, a vector of the row or time of each replicate can be entered. The function assumes each replicate ends at the row preceding the start of the next replicate, or for the final replicate the final row of the dataset.
`wait`	Numeric. Rows or time period to exclude at the start of each replicate. Default is `NULL` in which case no wait phase is applied. See Details.
`measure`	Numeric. Rows or time period over which to calculate rate in each replicate. Applied directly after `wait` phase. Default is `NULL` in which case the entire replicate is used. See Details.
`by`	String. `"row"` or `"time"`. Defaults to `"row"`. Method by which `starts`, `wait` and `measure` are applied.
`plot`	Logical. Default is `TRUE`. Plots the results. See 'Plotting' section for details.
`...`	Allows additional plotting controls to be passed, such as `type`, `pos`, `legend`, and `quiet`.

Details

`calc_rate.int` uses the `starts` input to subset each replicate. The `wait` and `measure` inputs control which parts of each replicate data are excluded and included from the rate calculation. It extracts a rate from each replicate using these, and saves it and other data to a summary table.

The `x` input should be an`inspect` object. Alternatively, it can be a two-column data frame containing paired values of time and oxygen from an intermittent-flow experiment in columns 1 and 2 respectively (though we always recommend processing such data in `inspect()` first). If a multiple column dataset is entered as `x` the first two columns are selected by default. If these are not the intended data use `inspect` to select the correct time and oxygen columns.

Specifying replicate structure:

The `starts` input specifies the locations of the start of each replicate in the data in `x`. This can be in one of two ways:

- A single numeric value specifying how replicates are spaced starting from the data in the first row. This option should only be used when replicates cycle at regular intervals. This can be a regular row or time interval, as specified via the `by` input. If the first replicate does not start at row 1, the data should be subset so that it does (see `subset_data()`) and example here.
- A numeric vector of row locations or times, as specified via the `by` input, of the start of each individual replicate. The first replicate does not have to start at the first row of the data, and all data after the last entry is assumed to be part of the final replicate. Regular `R` syntax such as `seq()`, `1:10`, etc. is also accepted, so can be used to specify both regular and irregular replicate spacing.

For both methods it is assumed each replicate ends at the row preceding the start of the next replicate, or in the case of the last replicate the final row of the dataset.

Specifying rate region:

The `wait` and `measure` inputs are used to specify the region from which to extract a rate and exclude flush periods. They can be entered as row intervals or time values in the units of the input data. The `wait` phase controls the amount of data at the start of each replicate to be ignored, that is not used in rate calculations. The `measure` phase determines the region after this over which a rate is calculated. There is no `flush` phase input since this is assumed to be from the end of the `measure` phase to the end of the replicate.

Both `wait` and `measure` can be entered in one of two ways:

- Single numeric values specifying a row width or a time period, as specified via the `by` input. Use this if you want to use the *same* `wait` and `measure` phases in every replicate, that is extract a rate from the same region of each.
- If `starts` is a vector of locations of the start of each replicate, these inputs can also be vectors of equal length of row lengths or time periods as specified via the `by` input. This is only useful if you want to use *different* `wait` and/or `measure` phases in different replicates.

If `wait = NULL` no wait phase is applied. If `measure = NULL` the rate is extracted from the start of the replicate or end of the `wait` phase to the last row of the replicate. This will typically include the flush period, so is rarely what you would want. Similarly if any `measure` input is beyond the available values in the replicate the closest value (row or time) is used instead, which again would typically be the last row of the replicate.

Example:

See examples below for actual code, but here is a simple example. An experiment comprises replicates which cycle at ten minute intervals with data recorded every second. Therefore each replicate will be 600 rows long. Flushes of the respirometer take 3 minutes at the end of each replicate. We want to exclude the first 2 minutes (120 rows) of data in each, and measure an oxygen uptake rate for five minutes (300 rows), leaving the three minutes of flushing (180 rows) excluded. The inputs for this would be:
`starts = 600, wait = 120, measure = 300, by = "row"`

More details:

Only a single rate can be extracted from each replicate. If for some reason you need to extract multiple rates from single replicates use `subset_data()` and `calc_rate()` which accepts multiple `from` and `to` inputs. Similarly, the `calc_rate` method of `by = "oxygen"` is not supported in `calc_rate.int`. See vignettes on the website for examples of alternative ways of iterating `calc_rate` across multiple replicates if you need to get around these constraints.

Plot:

If `plot = TRUE` (the default), the result for each replicate is plotted on a grid up to a maximum of 20. Which replicates are plotted can be selected using the pos input (default is 1:20), either in the main function call or when calling `plot()` on output objects.

There are three ways of plotting the results, which can be selected using the `type` input:

- `type = "rep"`: The default. Each individual replicate is plotted with the rate region (i.e. `measure` phase) highlighted in yellow. The `wait` and `measure` phases are also highlighted with red and green backgrounds respectively. These are also labelled if `legend = TRUE`.
- `type = "full"`: Each replicate rate (i.e. `measure` phase) is highlighted in the context of the whole dataset. May be quite difficult to interpret if dataset is large.
- `type = "cr"`: Plots individual replicate results as `calc_rate` objects.

For all types `pos` can be used to select which replicate(s) to plot.

S3 Generic Functions:

Saved output objects can be used in the generic S3 functions `plot()`, `print()`, `summary()`, and `mean()`.

- `plot()`: plots the result. See Plot section above.
- `print()`: prints the result of a single replicate, by default the first. Others can be printed by passing the pos input. e.g. `print(x, pos = 2)`
- `summary()`: prints summary table of all results and metadata, or the rows specified by the `pos` input. e.g. `summary(x, pos = 1:5)`. The `$rep` column indicates the replicate number. The summary table can be exported as a separate data frame by passing `export = TRUE`.
- `mean()`: calculates the mean of the rates from every replicate, or the rows specified by the `pos` input. e.g. `mean(x, pos = 1:5)` The mean can be exported as a numeric value by passing `export = TRUE`.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at `https://januarharianto.github.io/respR/`

Value

Output is a `list` object of class `calc_rate.int` containing a `calc_rate` object for each replicate in `$results`. The output also contains a `$summary` table which includes the full rate regression results from each replicate with replicate number indicated by the `$rep` column. Output also contains a `$rate` element which contains the rate values from each replicate in order. The function call, inputs, and other metadata are also included. Note, that if you have many replicates this object can be rather large (several MB).

Examples

```
# Irregular replicate structure ------------------------------------------

# Prepare the data to use in examples
# Note in this dataset each replicate is a different length!
data <- intermittent.rd
# Convert time to minutes (to show different options below)
data[[1]] <- round(data[[1]]/60, 2)
# Inspect
urch_insp <- inspect(data)

# Calculate rate across each entire replicate
# This leads to erroneous rates because the flush is included
calc_rate.int(urch_insp,
              starts = c(1, 2101, 3901))

# So instead we also specify a 'measure' phase
calc_rate.int(urch_insp,
              starts = c(1, 2101, 3901),
              measure = 1000)

# You can even specify different 'measure' phases in each rep
calc_rate.int(urch_insp,
              starts = c(1, 2101, 3901),
              measure = c(1500, 1200, 200))

# We usually don't want to use the start of a replicate just after the flush,
# so we specify a 'wait' phase. We can also specify 'starts', 'wait' and
# 'measure' in units of time instead of rows.
#
# By time
# (this time we save the result)
urch_res <- calc_rate.int(urch_insp,
                          starts = c(0, 35, 65), # start locations in minutes
                          wait = 2,              # wait for 2 mins
                          measure = 10,          # measure for 10 mins
                          by = "time")

# Regular replicate structure --------------------------------------------

# If replicates cycle at regular intervals, 'starts' can be used to specify
# the spacing in rows or time, starting at row 1. Therefore data must be
# subset first so that the first replicate starts at row 1.
#
# Subset and inspect data
zeb_insp <- zeb_intermittent.rd %>%
  subset_data(from = 5840,
              to = 75139,
              by = "row",
              quiet = TRUE) %>%
  inspect()

# Calculate a rate from same 6-minute region in every replicate.
# Replicates cycle at every 660 rows.
zeb_res <- calc_rate.int(zeb_insp,
                         starts = 660,
                         wait = 120, # exclude first 2 mins
                         measure = 360, # rate from 6 mins after 'wait'
                         plot = TRUE)

# S3 functions ------------------------------------------------------------

# Outputs can be used in print(), summary(), and mean().
# 'pos' can be used to select replicate ranges
summary(zeb_res)
mean(zeb_res, pos = 1:5)

# There are three ways by which the results can be plotted.
# 'pos' can be used to select replicates to be plotted.
#
# type = "rep" - the default. Each replicate plotted on a grid with rate
# region highlighted (up to a maximum of 20).
plot(urch_res)

# type = "full" - each replicate rate region plotted on entire data series.
plot(urch_res, pos = 1:2, type = "full")
```

```
# Of limited utility when datset is large
plot(zeb_res, pos = 10, type = "full")

# type = "cr" - the 'calc_rate' object for selected replicates in 'pos' is plotted
plot(urch_res, pos = 2, type = "cr")

# See vignettes on website for how to adjust and convert rates from calc_rate.int
```

`convert_DO` *Convert between units of dissolved oxygen*

Description

This is a conversion function that performs conversions between concentration and pressure units of dissolved oxygen (DO).

Usage

```
convert_DO(
  x,
  from = NULL,
  to = NULL,
  S = NULL,
  t = NULL,
  P = NULL,
  simplify = TRUE
)
```

Arguments

`x`	numeric. The dissolved oxygen (DO) value(s) to be converted.
`from`	string. The DO unit to convert *from*. See `unit_args()` for details.
`to`	string. The DO unit to convert *to*. See `unit_args()` for details.
`S`	numeric. Salinity (ppt). Defaults to NULL. Required for conversion of some units. See `unit_args()` for details.
`t`	numeric. Temperature(°C). Defaults to NULL. Required for conversion of some units. See `unit_args()` for details.
`P`	numeric. Pressure (bar). Defaults to 1.013253. Required for conversion of some units. See `unit_args()` for details.
`simplify`	logical. Defaults to `TRUE` in which case the converted values are returned as a numeric vector. if `FALSE` a list object of class `convert_DO` is returned.

Details

The function uses a fuzzy string matching algorithm to accept various unit formatting styles. For example, `"mg/l"`, `"mg/L"`, `"mgL-1"`, `"mg l-1"`, `"mg.l-1"` are all parsed the same. See `[unit_args()]` for details of accepted units.

Oxygen concentration units should use SI units (`L` or `kg`) for the denominator.

Some DO units require temperature (`t`), salinity (`S`), and atmospheric pressure (`P`) to be specified; if this is the case the function will stop and prompt for them. For the atmospheric pressure input (P), a default value of 1.013 bar (standard pressure at sea level) is applied if not otherwise entered. For freshwater experiments, salinity should be set to zero (i.e. `S = 0`).

S3 Generic Functions:

Saved output objects (if `simplify = FALSE` is used) can be entered in the generic S3 functions `print()` and `summary()`.

- `print()`: prints input and converted values (up to first 20), plus input and output units.
- `summary()`: simple wrapper for `print()` function. See above.

Value

By default (`simplify = TRUE`) the output is a numeric vector of converted values. If `simplify = FALSE` output is a `list` object of class `convert_DO` containing five elements: `$call` the function call, `$input` values, `$output` converted values, `$input.unit` and `$output.unit`.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at https://januarharianto.github.io/respR/

Examples

```
# Convert a numeric value to units which do not require t, S and P
convert_DO(8.21, from = "mg/L", to = "umol/L")

# Convert a numeric value to units which require t, S and P
convert_DO(100, from = "%Air", to = "mg L-1", S = 33, t = 18)
convert_DO(214, from = "hPa", to = "mL/kg", S = 33, t = 18)

# Convert a vector of values
convert_DO(urchins.rd[[5]], from = "mg/L", to = "umol/L")
convert_DO(c(8.01, 8.03, 8.05), from = "mg per litre", to = "%Air",
  t = 15, S = 35)
convert_DO(sardine.rd[[2]], from = "%Air", to = "torr",
  t = 15, S = 35)
```

convert_MR — *Convert between units of absolute, mass-specific, or area-specific metabolic rates*

Description

Converts metabolic rates to a different unit. These can be absolute rates (i.e. whole chamber or whole specimen e.g. `mg/h`), mass-specific rates (i.e. normalised by specimen mass e.g. `mg/h/kg`), or area-specific rates (i.e. normalised by specimen surface area e.g. `mg/h/cm2`). Input rates can be a numeric value or vector, in which case the converted rates are output as a numeric of the same length in the new units. Alternatively, input can be an object of class `convert_rate()` or `convert_rate.ft()`, in which case a new `convert_rate` or `convert_rate.ft` object is returned with all rates in the `$summary` and other elements converted to the new units. This allows you to convert results of analyses to a different unit without having to repeat the entire analysis.

Usage

```
convert_MR(
  x,
  from = NULL,
  to = NULL,
  S = NULL,
  t = NULL,
  P = NULL,
  quiet = FALSE
)
```

Arguments

`x`	numeric value or vector, or object of class `convert_rate` or `convert_rate.ft`. Contains the rate(s) to be converted.
`from`	string. The unit of the input metabolic rate(s). Should be in the correct order: Oxygen/Time, Oxygen/Time/Mass or Oxygen/Time/Area. If `x` is a `convert_rate` or `convert_rate.ft` object this is extracted automatically and any input specified via `from` is ignored.
`to`	string. The unit to convert the metabolic rate(s) to. Should be in the correct order: Oxygen/Time, Oxygen/Time/Mass or Oxygen/Time/Area. See `unit_args()`.
`S`	numeric. Salinity (ppt). Defaults to NULL. Used in conversion of some oxygen units. Freshwater should be entered as `S = 0`. If `x` is a `convert_rate` or `convert_rate.ft` object this is extracted automatically if present and any other input ignored.
`t`	numeric. Temperature(°C). Defaults to NULL. Used in conversion of some oxygen units. If `x` is a `convert_rate` or `convert_rate.ft` object this is extracted automatically if present and any other input ignored.

`P`	numeric. Pressure (bar). Used in conversion of some oxygen units. Defaults to a standard value of 1.013253 bar. If `x` is a `convert_rate` or `convert_rate.ft` object this is extracted automatically if present and any other input ignored.
`quiet`	logical. Suppresses the summary of the converted rates printed to the console. Default is `FALSE`.

Details

Units:

Units are specified using `from` and `to`. These should be in the sequence *Oxygen-Time* (e.g. `"mg/h"`) for absolute rates, *Oxygen-Time-Mass* (e.g. `"mg/h/kg"`) for mass-specific rates, and *Oxygen-Time-Area* (e.g. `"mg/h/cm2"`) for surface area-specific rates. If `x` is a `convert_rate` or `convert_rate.ft` object the `from` unit is extracted automatically.

A fuzzy string matching algorithm is used to accept various unit formatting styles. For example, `"mg/h"`, `"mg/H"`, `"mg hr-1"`, `"milligram per hour"` are all parsed the same. See `unit_args()` for details of accepted units and their formatting.

Note some units require salinity (`S`) and temperature (`t`) to perform the conversion. For freshwater experiments, salinity should be entered as zero (i.e. `S = 0`). These conversions also require the atmospheric pressure (`P`). If not entered the default value of 1.013253 bar (standard pressure at sea level) is used. If `x` is a `convert_rate` or `convert_rate.ft` object, `S`, `t`, and `P` are extracted automatically if they are present (they may not be if the original rate conversion did not require them). They are also saved to the `$inputs` element of the output object.

More details:

For `convert_rate` or `convert_rate.ft` inputs the primary `$rate.output` element is converted to the new unit and the `$output.unit` also updated to this new unit. These columns are also updated in the `$summary` table, and in addition the `rate.abs` column and, if relevant, the `rate.m.spec` or `rate.a.spec` column. Note, the `$call` element is updated to the `convert_MR` call and the original call to `convert_rate` or `convert_rate.ft` replaced. The `$inputs` element will still contain the original inputs, with the `output.unit` updated to the new `to` unit. In addition `S`, `t`, and `P` are added if they weren't already present.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at `https://januarharianto.github.io/respR/`

Value

If the `x` input rates are a numeric value or vector, output is a numeric value or vector of the same length. If `x` is a `convert_rate` or `convert_rate.ft` object, output is a new `convert_rate` or `convert_rate.ft` object with all rates in the `$summary` table, `$rate.output` and `$output.unit` elements converted to the new units. This allows you to convert results of analyses to a different unit without having to repeat the entire analysis.

Examples

```
# Convert a numeric absolute rate to a different unit
convert_MR(-0.09, from = 'mg/min', to = 'umol/hr')
```

```
# Convert a vector of absolute rates to a different unit
convert_MR(c(-0.090, -0.081, -0.098),
           from = 'mg/min', to = 'umol/hr')

# Convert to a unit which requires S, t, & P
convert_MR(-0.09, from = 'mg/min', to = 'ml/hour',
           S = 0, t = 20, P = 1.01)

# Convert mass-specific rates
convert_MR(-0.09, from = 'mg/min/g', to = 'ml/hour/kg',
           S = 0, t = 20, P = 1.01)

# Convert area-specific rates
convert_MR(-0.09, from = 'mg/min/mm2', to = 'ml/hour/cm2',
           S = 0, t = 20, P = 1.01)

# Convert from units largely only used in older papers.
# E.g. cubic cm (e.g. Tang 1933, Head 1962)
convert_MR(0.1, from = 'cc/hr/gm', to = 'mg/hr/g',
           S = 30, t = 20, P = 1.01)
convert_MR(0.6, from = 'cm3/hr', to = 'mg/hr',
           S = 28, t = 12, P = 1.01)
# uL (e.g. Zeuthen 1953, Newell & Northcroft 1967)
convert_MR(400, from = 'ul/hr', to = 'mg/hr',
           S = 30, t = 15, P = 1.01)
convert_MR(0.5, from = 'ul/hr/mg', to = 'mg/hr/g',
           S = 0, t = 20, P = 1.01)
# mm3 (e.g. Newell & Roy 1973)
convert_MR(1.5, from = 'mm3/hr', to = 'mg/hr',
           S = 30, t = 15, P = 1.01)

# Convert rates in a 'convert_rate' object. This avoids having to repeat
# an entire analysis to see the output in different units.

# Make a convert_rate object
cnv_rt.obj <- urchins.rd %>%
  auto_rate() %>%
  convert_rate(oxy.unit = "mg/L",
               time.unit = "min",
               output.unit = "mg/hr/kg",
               volume = 1,
               mass = 0.05)

# Now convert all results to "umol/min/g".
# The 'from' units are identified automatically from the object.
cnv_rt.obj.new <- convert_MR(cnv_rt.obj,
                             to = "umol/min/g")

# Compare the two:
summary(cnv_rt.obj)
summary(cnv_rt.obj.new)
```

convert_rate	*Convert a unitless oxygen rate value to absolute, mass-specific or area-specific rate*

Description

Converts a unitless rate derived from `calc_rate()`, `calc_rate.int()`, `auto_rate()`, `auto_rate.int()`, `adjust_rate()`, or `calc_rate.bg()` into an absolute rate (i.e. whole chamber or whole specimen), or mass-specific rate (i.e. normalised by specimen mass), or area-specific rate (i.e. normalised by specimen surface area) in any common unit.

Usage

```
convert_rate(
  x,
  oxy.unit = NULL,
  time.unit = NULL,
  output.unit = NULL,
  volume = NULL,
  mass = NULL,
  area = NULL,
  S = NULL,
  t = NULL,
  P = NULL,
  plot = FALSE,
  ...
)
```

Arguments

`x`	numeric value or vector, or object of class `calc_rate`, `calc_rate.int`, `auto_rate`, `auto_rate.int`, `adjust_rate`, or `calc_rate.bg`. Contains the rate(s) to be converted.
`oxy.unit`	string. The dissolved oxygen unit of the original raw data used to determine the rates in `x`.
`time.unit`	string. The time unit of the original raw data used to determine the rates in `x`.
`output.unit`	string. The output units to convert the input rates to. Should be in the correct order: "Oxygen/Time" or "Oxygen/Time/Mass" or "Oxygen/Time/Area".
`volume`	numeric. Volume of water in **litres** in the respirometer or respirometer loop.
`mass`	numeric. Mass/weight in **kg**. This is the mass of the specimen if you wish to calculate mass-specific rates.
`area`	numeric. Surface area in **m^2**. This is the surface area of the specimen if you wish to calculate surface area-specific rates.
`S`	numeric. Salinity (ppt). Defaults to NULL. Used in conversion of some oxygen units. Freshwater should be entered as `S = 0`.

`t`	numeric. Temperature(°C). Defaults to NULL. Used in conversion of some oxygen units.
`P`	numeric. Pressure (bar). Used in conversion of some oxygen units. Defaults to a standard value of 1.013253 bar.
`plot`	logical. Default is `FALSE`. Controls if a plot is produced. See Plot section.
`...`	Allows additional plotting controls to be passed. See Plot section.

Details

By default, `convert_rate` converts the primary `$rate` element from `calc_rate`, `calc_rate.int`, `auto_rate` and , `auto_rate.int` objects, the `$rate.adjusted` from `adjust_rate` objects, and the `$rate.bg` from `calc_rate.bg` objects. Additionally, any numeric value or vector of rates can be input as `x`.

Respirometer volume:

The `volume` of the respirometer is required and should be in litres (`L`). Note, the `volume` represents the *effective volume* of the respirometer, that is *volume of water* in the respirometry chamber. This is not necessarily the same as the volume of the respirometer. Typically, it is the volume of the respirometer *minus* the volume of the specimen. See here for help with calculating effective volumes. It also does not refer to the specimen volume.

Units:

The `oxy.unit` of the original raw data used to calculate the rate is required. Concentration units should use only SI units (`L` or `kg`) for the denominator, e.g. `"mg/L"`, `"mmol/kg"`. Percentage saturation of air (`%Air`) or oxygen (`%Oxy`) is supported, as are oxygen pressure units. See `unit_args()` for details.

The `time.unit` of the original raw data used to calculate the rate is also required (seconds, minutes, hours, or days).

An `output.unit` is also required and must be in the sequence *Oxygen-Time* (e.g. `"mg/h"`) for absolute rates, *Oxygen-Time-Mass* (e.g. `"mg/h/kg"`) for mass-specific rates, and *Oxygen-Time-Area* (e.g. `"mg/h/cm2"`) for surface area-specific rates. If left `NULL`, the default of `"mgO2/h"` is used, or `"mgO2/h/kg"` or `"mgO2/h/m2"` if a `mass` or `area` respectively has been entered.

Note, some oxygen input or output units require temperature (`t`) and salinity (`S`) to perform conversions. For freshwater experiments, salinity should be entered as zero (i.e. `S = 0`).

Strictly speaking, the atmospheric pressure (`P`) should also be entered. If not, the default value of 1.013253 bar (standard pressure at sea level) is used. In most locations which have a normal range of around 20 millibars (outside of extreme weather events), any variability in pressure will have a relatively minor effect on dissolved oxygen, and even less on calculated rates. However, we would encourage users to enter the actual value if they know it, or use historical weather data to find out what it was on the day. See `unit_args()` for details.

The function uses an internal database and a fuzzy string matching algorithm to accept various unit formatting styles. For example, `"mg/l"`, `"mg/L"`, `"mgL-1"`, `"mg l-1"`, `"mg.l-1"` are all parsed the same. See `unit_args()` for details of accepted units and their formatting. See also `convert_val()` for simple conversion between non-oxygen units.

Plot:

Plotting provides three ways of visualising the rates (or a selection of them using pos), chosen using type. The default is plot = FALSE to prevent plots being produced for every single conversion.

type = "full" (the default) plots a grid of up to 20 plots with each rate highlighted on the full dataset, with the rate value in the title. Values on the axes - time (bottom), row (top), and oxygen (left) - are in the units of the original raw data. Rates are plotted in order of how they appear in the summary table up to the first 20 rows, unless different rows have been specified via pos.

type = "rate" plots the entire data timeseries on the upper plot, and on the lower plot the output rate values in the chosen output units. Each rate is plotted against the middle of the region used to determine it. pos can be used to select a range of rates (i.e. summary table rows) to show in the lower plot (default is all).

type = "overlap" visualises where regression results in the summary table occur in relation to the original dataset to help understand how they are distributed or may overlap, and is particularly useful for results from the auto_rate linear method. The top plot is the entire data timeseries, the bottom plot the region of the data each rate regression has been fit over. The y-axis represents the position (i.e. row) of each in the summary table descending from top to bottom. If no reordering or selection has been performed, this will usually be equivalent to the $rank column, but note as reordering or selection is performed rank and summary table position will not necessarily be equivalent. One result (summary table row) can be highlighted, the default being highlight = 1. pos can be used to select a range of summary rows to plot in the lower overlap plot.

Other options:

legend = FALSE will suppress plot labels, pos selects summary rates to plot, quiet suppresses console messages.

S3 Generic Functions:

Saved output objects can be used in the generic S3 functions print(), summary(), and mean().

- print(): prints a single result, by default the first converted rate. Others can be printed by passing the pos input. e.g. print(x, pos = 2)
- summary(): prints the output $summary table of converted rates and metadata. Specific rows can be specified with the pos input. e.g. summary(x, pos = 1:5). This can be exported as a separate data frame by passing export = TRUE and includes all rate regression parameters, and data locations, adjustments if applied, units, and more. The $rep and $rank columns requires special notice depending on the type of experiment you have analysed or the function you used to determine the rates. For the $rank column if calc_rate was used, it is the order of rates as entered using from and to (if multiple rates were determined). For auto_rate it relates to the method input, for example it indicates kernel density ranking if the linear method was used, or ordering by rate value if lowest or highest were used. For intermittent-flow experiments analysed via calc_rate.int or auto_rate.int it indicates the ranking *within* each replicate as seen in the $rep column. Note that if select_rate has been used the rows in the summary table may have been reordered, including the $rep and $rank columns. The *original* rep and rank for each row is retained if reordering occurred.
- mean(): calculates the mean of all converted rates, or those specified by the pos input. e.g. mean(x, pos = 1:5) The mean can be exported as a separate value by passing export = TRUE.

More:

For additional help, documentation, vignettes, and more visit the respR website at https://januarharianto.github.io/respR/

Value

Output is a `list` object of class `convert_rate` containing the `$rate.input`, and converted rate(s) in `$rate.output` in the `$output.unit`, as well as inputs and summary elements. Note, `$rate.abs` is the *absolute* rate in the output unit minus the mass- or area-specific component. The `$summary` table element contains all rate regression parameters and data locations (depending on what class of object was entered), adjustments (if applied), units, and more. The `$rep` and `$rank` columns require special notice depending on the type of experiment you have analysed or the function you used to determine the rates. See the summary table description in **S3 Generic Functions** section above.

Examples

```
# Convert a single numeric rate to an absolute rate
convert_rate(0.09, oxy.unit = 'mg/l', time.unit = 's',
  output.unit = 'mg/min', volume = 1.2)

# Convert a single numeric rate to a mass-specific rate
convert_rate(0.09, oxy.unit = 'mg/l', time.unit = 's',
  output.unit = 'mg/min/kg', volume = 1.2, mass = 0.5)

# Convert a single numeric rate to an area-specific rate
convert_rate(0.09, oxy.unit = 'mg/l', time.unit = 's',
  output.unit = 'mg/min/cm2', volume = 1.2, area = 0.0002)

# Convert a single rate derived via calc_rate to mass-specific
x <- calc_rate(sardine.rd, from = 200, to = 1800, by = "time")
convert_rate(x, oxy.unit = '%Air', time.unit = 's',
  output.unit = 'mg/h/g', volume = 12.3, mass = 0.05,
  S =35, t = 15, P = 1.013)

# Convert multiple rates derived via auto_rate to area-specific
x <- auto_rate(sardine.rd)
rates <- convert_rate(x, oxy.unit = '%Air', time.unit = 's',
  output.unit = 'mg/h/cm2', volume = 12.3, area = 0.00005,
  S =35, t = 15, P = 1.013)
summary(rates)
```

`convert_rate.ft` *Convert a unitless oxygen rate value from flowthrough respirometry to absolute, mass-specific or area-specific rates*

Description

`convert_rate.ft` converts a unitless rate derived from `calc_rate.ft()` or `adjust_rate.ft()` into an absolute rate (i.e. whole specimen or whole chamber), mass-specific rate (i.e. normalised by specimen mass), or area-specific rate (i.e. normalised by specimen surface area) in any common unit. These should be rates calculated as an oxygen delta (inflow minus outflow oxygen) multiplied by the flowrate.

Usage

```
convert_rate.ft(
  x,
  oxy.unit = NULL,
  flowrate.unit = NULL,
  output.unit = NULL,
  mass = NULL,
  area = NULL,
  S = NULL,
  t = NULL,
  P = 1.013253,
  plot = FALSE,
  ...
)
```

Arguments

`x`	numeric value or vector, or object of class `calc_rate.ft()` or `adjust_rate.ft()`. Contains the rate(s) to be converted.
`oxy.unit`	string. The dissolved oxygen units of the original raw data used to determine the rate in `x`.
`flowrate.unit`	string. The units of the flowrate through the respirometer. See Details.
`output.unit`	string. The output unit to convert the input rate to. Should be in the correct order: "Oxygen/Time" or "Oxygen/Time/Mass" or "Oxygen/Time/Area".
`mass`	numeric. Mass/weight in **kg**. This is the mass of the specimen if you wish to calculate mass-specific rates.
`area`	numeric. Surface area in **m^2**. This is the surface area of the specimen if you wish to calculate surface area-specific rates.
`S`	numeric. Salinity (ppt). Defaults to NULL. Used in conversion of some oxygen units. Fresh water should be entered as `S = 0`.
`t`	numeric. Temperature(°C). Defaults to NULL. Used in conversion of some oxygen units.
`P`	numeric. Pressure (bar). Used in conversion of some oxygen units. Defaults to a standard value of 1.013253 bar.
`plot`	logical. Default is `FALSE`. Controls if a plot is produced. See Plot section.
`...`	Allows additional plotting controls to be passed. See Plot section.

Details

By default, `convert_rate.ft` converts the `$rate` element from `calc_rate.ft` objects, or the `$rate.adjusted` element from `adjust_rate.ft` objects if these are entered as the `x` input. Alternatively, a numeric value or vector of rates can be input as `x`.

Units:

The `oxy.unit` of the original raw data used to calculated the rate is required. Concentration units should use only SI units (`L` or `kg`) for the denominator, e.g. `"mg/L"`, `"mmol/kg"`. Percentage saturation of air or oxygen is accepted, as are oxygen pressure units. See `unit_args()` for details.

An `output.unit` is also required. If left `NULL`, The default of `"mgO2/h"` is used, or `"mgO2/h/kg"` or `"mgO2/h/m2"` if a `mass` or `area` respectively has been entered. The `output.unit` must be in the sequence *Oxygen-Time* (e.g. `"mg/h"`) for absolute rates, *Oxygen-Time-Mass* (e.g. `"mg/h/kg"`) for mass-specific rates, and *Oxygen-Time-Area* (e.g. `"mg/h/cm2"`) for surface area-specific rates.

Note, some oxygen input or output units require temperature (`t`) and salinity (`S`) to perform conversions. For freshwater experiments, salinity should be entered as zero (i.e. `S = 0`).

Strictly speaking the atmospheric pressure (`P`) should also be supplied. If not, the default value of 1.013253 bar (standard pressure at sea level) is used. In most locations which have a normal range (outside extreme weather events) of around 20 millibars, any variability in pressure will have a relatively minor effect on dissolved oxygen, and even less on calculated rates. However, we would encourage users to enter the actual value if they know it, or use historical weather data to find out what it was on the day. See `unit_args()` for details.

The `flowrate.unit` is required and should be the units of the `flowrate` used in `calc_rate.ft` to calculate the rate, and should be in the form of volume (L, ml, or ul) per unit time (s,m,h,d), for example in `"L/s"`. Note, the volume component does *NOT* represent the volume of the respirometer, and the time component does *NOT* represent the units or recording interval of the original raw data.

The function uses a fuzzy string matching algorithm to accept various unit formatting styles. For example, `"mg/l"`, `"mg/L"`, `"mgL-1"`, `"mg l-1"`, `"mg.l-1"` are all parsed the same. See `unit_args()` for details of accepted units and their formatting. See also `convert_val()` for simple conversion between non-oxygen units.

Plot:

Plotting provides three ways of visualising converted rates (or a selection of them using `pos`), chosen using `type`. This is mostly useful only if you have extracted multiple rates (see `calc_rate.ft()`). The default is `plot = FALSE` to prevent plots being produced for every single conversion. `convert_rate.ft` objects can only be plotted if and `inspect.ft` object was used as the input in `calc_rate.ft`. In other words, converted rates from numeric inputs cannot be plotted.

`type = "full"` (the default) plots a grid of up to 20 plots with each rate (i.e. region of averaged delta values) highlighted on a plot of delta oxygen values, with the converted rate value in the title. Values on the axes - time (bottom), row (top), and oxygen delta (left) - are in the units of the original raw data. Rates are plotted in order of how they appear in the summary table up to the first 20 rows, unless different rows have been specified via `pos`.

`type = "rate"` plots the entire data timeseries, that is the outflow and inflow oxygen (if used) on the upper plot, with delta oxygen on the middle plot or as the upper plot if delta oxygen values have been entered in `inspect.ft`. The lower plot is the output rate values in the chosen output units. Each rate is plotted against the middle of the region used to determine it (i.e. region of averaged delta values). `pos` can be used to select a range of rates (i.e. summary table rows) to show in the lower plot (default is all).

`type = "overlap"` visualises where regression results in the summary table occur in relation to the original dataset to help understand how they are distributed or may overlap. The top plot is the entire data timeseries, that is the outflow and inflow oxygen (if used) on the upper plot, with delta oxygen on the middle plot or as the upper plot if delta oxygen values have been entered in `inspect.ft`. The bottom plot is the region of the data each rate has been calculated over (i.e. region of averaged delta values). The y-axis represents the position (i.e. row) of each in the summary table descending from top to bottom. If no reordering or selection has been performed, this will usually be equivalent to the `$rank` column, but note as reordering or selection is performed rank and summary table position will not necessarily be equivalent. One result (summary table row) can be highlighted, the default being `highlight = 1`. `pos` can be used to select a range of summary rows to plot in the lower overlap plot.

Other options:

`legend = FALSE` will suppress plot labels, `pos` selects summary rates to plot, `quiet` suppresses console messages.

S3 Generic Functions:

Saved output objects can be used in the generic S3 functions `print()`, `summary()`, and `mean()`.

- `print()`: prints a single result, by default the first converted rate. Others can be printed by passing the `pos` input. e.g. `print(x, pos = 2)`
- `summary()`: prints the output `$summary` table of converted rates and metadata. Specific rows can be specified with the `pos` input. e.g. `summary(x, pos = 1:5)`. This can be exported as a separate data frame by passing `export = TRUE`, and includes all rate parameters, data locations, adjustments if applied, units, and more. Note, the summary table contains linear regression coefficients alongside other metadata. These should not be confused with those in other functions such as `calc_rate` where slopes represent rates and coefficients such as a high r-squared are important. Here, slope represents the stability of the data region, in that the closer the slope is to zero, the less the delta oxygen values in that region vary, which is an indication of a region of stable rates. They are included to enable possible future functionality where stable regions may be automatically identified, and should generally be ignored. However, advanced users can use regular R syntax to explore and subset the results using these if they wish.
- `mean()`: calculates the mean of all converted rates, or those specified by the `pos` input. e.g. `mean(x, pos = 1:5)` The mean can be exported as a separate value by passing `export = TRUE`.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at `https://januarharianto.github.io/respR/`

Value

Output is a `list` object containing the `$rate.input`, and converted rate(s) in `$rate.output` in the `$output.unit`, as well as inputs and summary elements. Note, `$rate.abs` is the *absolute* rate in the output unit minus the mass- or area-specific component. The `$summary` table element contains all rate parameters and data locations (depending on what class of object was entered), adjustments (if applied), units, and more.

Examples

```
# Convert a single numeric rate to an absolute rate
convert_rate.ft(-0.09, oxy.unit = 'mg/l', flowrate.unit = 'L/s',
                output.unit = 'mg/min')

# Convert a single numeric rate to a mass-specific rate
```

```
convert_rate.ft(-0.09, oxy.unit = 'mg/l', flowrate.unit = 'L/s',
                output.unit = 'mg/min/kg', mass = 0.5)

# Convert a single numeric rate to an area-specific rate
convert_rate.ft(-0.09, oxy.unit = 'mg/l', flowrate.unit = 'L/s',
                output.unit = 'mg/min/cm2', area = 0.0002)

# Full object-oriented workflow
# Inspect, calculate rate, adjust rate, and convert
# to a final mass-specific rate
inspect.ft(flowthrough_mult.rd,
           time = 1,
           out.oxy = 2,
           in.oxy = 6) %>%
  calc_rate.ft(flowrate = 0.1,
               from = 30,
               to = 60,
               by = "time") %>%
  adjust_rate.ft(by = -0.032) %>%
  convert_rate.ft(oxy.unit = '%Air',
                  flowrate.unit = 'L/min',
                  output.unit = 'mg/h/g',
                  mass = 0.05,
                  S =35, t = 15, P = 1.013)
```

convert_val — *Convert values of temperature, volume, mass, area, and atmospheric pressure to different units*

Description

This is a basic function that converts values of temperature, volume, mass, area, and atmospheric pressure to different units. This can be useful in `convert_DO()`, `convert_rate()`, and `convert_rate.ft()` where some inputs must be in specific units (e.g. temperature in °C, atmospheric pressure in bar, area in m2). See Examples.

Usage

```
convert_val(x, from = NULL, to = NULL)
```

Arguments

`x`	numeric value or vector. Values to be converted to a different unit.
`from`	string. Unit of the original values.
`to`	string. Unit to be converted to. These defaults are applied if left `NULL`: volume `"L"`, temperature `"C"`, mass `"kg"`, area `"m2"`, pressure `"bar"`.

Details

Note the type of unit does not need to be specified. The function will automatically recognise it using the `from` unit.

If the `'to'` input is left `NULL`, the following defaults are applied depending on the unit type of the `from` input:

- volume: `"L"`
- temperature: `"C"`
- mass: `"kg"`
- area: `"m2"`
- pressure: `"bar"`

A fuzzy string matching algorithm is used to accept different unit formatting styles. For example, `"msq"` `"m2"`, `"M2"`, `"sqm"` are all parsed as metres squared of area.

Accepted Units:

Temperature:

- `"C"`, `"K"`, `"F"`

Pressure:

- `"kPa"`, `"hPa"`, `"Pa"`, `"ubar"`, `"mbar"`, `"bar"`, `"Torr"`, `"atm"` (note, this is standard atmospheres).

Volume:

- `"uL"`, `"mL"`, `"L"`

Mass:

- `"ug"`, `"mg"`, `"g"`, `"kg"`

Area:

- `"mm2"`, `"cm2"`, `"m2"`, `"km2"`

More:

For additional help, documentation, vignettes, and more visit the `respR` website at `https://januarharianto.github.io/respR/`

Value

Output is a numeric vector of converted values.

Examples

```
# Convert volume
convert_val(10, "ml", "L")
convert_val(10:15, "ml", "L")

# Convert temperature
convert_val(-273.15, "C", "K")
convert_val(-40, "C", "F")
convert_val(c(2,4,6,8), "C", "F")
```

```
# Convert pressure
convert_val(1, "atm", "bar")
convert_val(1010, "hpa", "bar")
convert_val(735, "torr", "kpa")

# Convert area
convert_val(100, "cm2", "m2")
convert_val(10000, "mm2", "cm2")

# Convert mass
convert_val(200, "g", "kg")
convert_val(10000, "ug", "mg")

# Use directly in a respR function which requires inputs to be
# in a specific unit. For example, in convert_rate() pressure
# must be in 'bar' and respirometer volume in 'L'.
# Here, we know chamber volume is 200 ml, and pressure measured in mbar.
x <- suppressWarnings(inspect(urchins.rd, 1, 2))

rate <- calc_rate(x, from = 20, to = 30)

convert_rate(rate,
             oxy.unit = "ml/l",
             time.unit = "min",
             output.unit = "mg/h",
             volume = convert_val(200, "ml", "L"),
             S = 35,
             t = 15,
             P = convert_val(1010, "mbar", "bar"))

# Note, the default 'to' units are set to those respR requires in
# these functions ('L' and 'bar' here), so do not necessarily need
# to be specified:
convert_rate(rate,
             oxy.unit = "ml/l",
             time.unit = "min",
             output.unit = "mg/h",
             volume = convert_val(200, "ml"),
             S = 35,
             t = 15,
             P = convert_val(1010, "mbar"))
```

`flowthrough.rd` *Flowthrough respirometry data on the chiton,* Mopalia lignosa

Description

A single experiment on the chiton species *Mopalia lignosa* in a custom-built flowthrough respirometry system. Conducted at University of British Columbia, Vancouver, BC, Canada.

Usage

```
flowthrough.rd
```

Format

A data frame object consisting of 935 rows (approx 16 mins of data),and 4 columns: time, oxygen inflow and outflow concentrations, and oxygen delta (the outflow minus inflow concentrations).

Details

- Dissolved oxygen units: `mg/L`
- Time units: `seconds`
- Flow rate (`mL/min`): `2.34`
- Inflow oxygen concentration (calculated assuming 100% air saturated, `mg/L`): `8.919`
- Specimen ash-free dry mass (kg): `0.000070`
- Temperature (°C): `t = 12`
- Salinity: `S = 30`
- Atm. Pressure (bar): `P = 1.013`

Author(s)

Nicholas Carey

`flowthrough_mult.rd` *Multi-column flowthrough respirometry data*

Description

A semi-simulated dataset for testing and demonstrating flowthrough respirometry analyses. Contains one column of numeric time data (col 1 in mins), four columns of outflow oxygen concentrations (cols 2:5), four columns of inflow oxygen concentrations (cols 6:9), and four columns of delta oxygen concentrations (cols 10:13, which is simply the numeric difference between paired columns of outflow and inflow). There is also a column of inflow oxygen concentrations as recorded from a shared header tank (col 14, `$oxy.header`) supplying all chambers, to use as an alternative to the individual inflow oxygen recordings. Lastly, there is a column of temperature data (col 15, `$temperature` in °C).

Usage

```
flowthrough_mult.rd
```

Format

A data frame object consisting of 3740 rows (approx 62 mins of data),and 15 columns: time (col 1), oxygen outflow concentrations (cols 2,3,4,5), inflow concentrations (cols 6,7,8,9 each paired with the respective outflow column, the fourth being a control), delta oxygen values (cols 10,11,12,13 or difference between outflow and inflow concentrations), inflow concentrations recorded in a shared header tank (col 14), and temperature (col 15).

Details

Outflow (2:5) and inflow (6:9) columns are paired, with the first three containing specimens, and the fourth an empty control respirometer, or "blank" experiment (oxy.out.blank, oxy.in.blank) to determine background respiration.

The third paired dataset (col 4 and col 8 pair) has a period of higher rates at around the 40 minute timepoint, where the specimen increases its activity then slowly recovers to routine respiration levels.

- Dissolved oxygen units: `%Air`
- Time units: `mins`
- Flow rate (`L/min`): `0.1`
- Specimen masses: (kg): `0.013, 0.015, 0.020`
- Mean temperature (°C): `t = 18`
- Salinity: `S = 0`, i.e. freshwater
- Atmospheric pressure (bar): `P = 1.013`

Author(s)

Nicholas Carey

`flowthrough_sim.rd` *Flowthrough respirometry data with increasing background rate*

Description

A simulated dataset for testing and demonstrating flowthrough respirometry analyses and background adjustment when the background respiration rate increases over the course of the experiment. Contains one column of numeric time data (`$num.time`), one column of specimen outflow oxygen concentrations (`$oxy.out.spec`), one column of control or "blank" chamber outflow oxygen concentrations (`$oxy.out.blank`), and one column of inflow oxygen concentrations as recorded from a shared header tank (`$oxy.header`) supplying both chambers.

Usage

```
flowthrough_sim.rd
```

Format

A data frame object consisting of 3740 rows (approx 62 mins of data),and 4 columns: time (col 1), specimen oxygen outflow concentrations (col 2), control/blank chamber oxygen outflow concentrations (col 3), and inflow concentrations recorded from a shared header tank (col 4).

Details

- Dissolved oxygen units: `mg/L`
- Time units: `seconds`

Author(s)

Nicholas Carey

`format_time` *Parse date-time data to numeric time for use in respR functions*

Description

A function to parse class POSIX.ct or text strings of date-time data to numeric time for use in `respR` functions.

Usage

```
format_time(x, time = 1, format = "ymdHMS", start = 1)
```

Arguments

`x`	vector or data frame containing strings or class POSIX.ct date-time data to be converted to numeric.
`time`	numeric value or vector. Specifies column(s) containing date-time data. Default is `1`.
`format`	string. Code describing structure of date-time data. See Details.
`start`	numeric. At what time (in seconds) should the formatted time data start? Default is `1`.

Details

Regardless of input, all data are parsed to numeric time data in seconds duration from the first entry starting at 1. If you want the times to start at a different time, a `start` value can be specified, in which case the series starts at that number (in seconds) and all subsequent times are shifted forward by the same amount.

Input:

Input can be a vector, or data frame. If a data frame, the column(s) of the date-time data are specified using the `time` input. By default the first column is assumed to contain the date-time data (i.e. `time = 1`).

If the date-time data is split over several columns (e.g. date in one column, time in another), multiple columns can be specified (e.g. `time = c(1,2)`). In this case, the `format` setting should reflect the correct order as entered in `time`.

Time only data:

Time-only data, that is times which lack an associated date, can also be parsed. Normally, parsing time-only data will cause problems when the times cross midnight (i.e. `00:00:00`). However, the function attempts to identify when this occurs and parse the data correctly.

Formatting:

See the `lubridate` package for more detail on acceptable formatting.

Date-time data can be unspaced or separated by any combination of spaces, forward slashes, hyphens, dots, commas, colons, semicolons, or underscores. E.g. all these are parsed as the same date-time: `"2010-02-28 13:10:23"`, `"20100228131023"`, `"2010,02/28 13.10;23"`, `"2010 02 28 13_10-23"`.

- Times can be in 24H or 12H with AM/PM
  E.g. "2010-02-28 13:10:23" or "2010-02-28 1:10:23 PM"
- Times without initial zero are parsed as 24H time
  E.g. "1:10:23" is same as "1:10:23 AM" or "01:10:23"
- AM/PM take precedence over 24H formatting for 01-12h
  E.g. "1:10:23 PM" and "01:10:23 PM" are both same as "13:10:23"
- However, 24H formatting for 13-24h takes precedence over AM/PM
  E.g. "13:10:23 AM" is identified as "1:10:23 PM" or "13:10:23"

Syntax of 'format' input:

Specify the order of year, month, day, and time in your date-time input.

`d` - Day of the month as decimal number (01–31 or 1–31).

`m` - Month of the year as decimal number (01–12 or 1–12).

`y` - Year (2010, 2001, 1989).

`H` - Hour as decimal number (00–24 or 0–24 or 00-12 (see `p`)).

`M` - Minute as decimal number (00–59 or 0–59).

`S` - Second as decimal number (00–59 or 0–59).

`p` - AM/PM indicator for 12-h date-time format (e.g. "01/12/2020 1:30:44 PM " would be `"dmyHMSp"`).

Specify the order using the `format` input, using separators or not (optional): `"dmyHMS"`; `"dmy_HMS"` and `"d m y H M S"` are all the same. See Examples.

Single experimental datasets should never span different time zones, so if a time zone is present it is ignored for the purposes of calculating numeric times.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at `https://januarharianto.github.io/respR/`

Value

Output: If the input is a vector, output is a vector of equal length containing the numeric time data. For data frame inputs, an identical data frame is returned, with a new column named `time_num` added as the **final** column.

See Also

`lubridate`

Examples

```
# Convert year-month-day hour-min-sec
x <- c("09-02-03 01:11:11", "09-02-03 02:11:11","09-02-03 02:25:11")
format_time(x)

# Convert day-month-year hour-min, and use a separator in the format
x <- c("03-02-09 01:11", "03-02-09 02:11","03-02-09 02:25")
format_time(x, format = "dmy_HM")

# Convert when AM/PM is present
x <- c("09-02-03 11:11:11 AM", "09-02-03 12:11:11 PM","09-02-03 01:25:11 PM")
# This is WRONG - the AM/PM indicator is missing
format_time(x, format = "dmyHMS")
# This is correct
format_time(x, format = "dmyHMSp")

# Convert dataframe with year-month-day hour-min-sec (ymdHMS default)
x <- data.frame(
  x = c("09-02-03 01:11:11", "09-02-03 02:11:11","09-02-03 02:25:11"),
  y = c(23, 34, 45))
format_time(x, time = 1)

# Convert dataframe with time in a different column and non-default format
x <- data.frame(
  x = c(23, 34, 45),
  y = c("09-02-2018 11:11:11 AM", "09-02-2018 12:11:11 PM","09-02-2018 01:25:11 PM"),
  z = c(56, 67, 78))
format_time(x, time = 2, format = "dmyHMSp")

# Convert dataframe with separate date and time columns, and times crossing midnight
x <- data.frame(
  w = c("09-02-18", "09-02-18","10-02-18"),
  x = c("22:11:11", "23:11:11","00:25:11"),
  y = c(23, 34, 45),
  z = c(56, 67, 78))
# Crosses midnight, but parses correctly even without dates
format_time(x, time = 2, format = "HMS")
# Include dates to double check
format_time(x, time = 1:2, format = "dmyHMS")
# Input same as different column order & appropriate format order
format_time(x, time = 2:1, format = "HMSdmy")
```

```
# Convert a data frame with date and time split over multiple columns
x <- data.frame(
  u = c("09", "09","10"),
  v = c("02", "02","02"),
  w = c("2018", "2018","2018"),
  x = c("22:11:11", "23:11:11","00:25:11"),
  y = c(23, 34, 45),
  z = c(56, 67, 78))
format_time(x, time = 1:4, format = "dmyHMS")
```

---

import_file *Import respirometry system raw data files (DEPRECATED)*

Description

Automatically import data from different respirometry hardware and software systems. *WARNING*: This function is now deprecated and will not be updated. See below.

Usage

```
import_file(path, export = FALSE)
```

Arguments

`path`	string. Path to file.
`export`	logical. If TRUE, exports the data as a `csv` to the same directory, as determined by the `path` parameter.

Details

This function has been deprecated, will not be updated, and will be removed entirely in the next major version of `respR` (i.e. `v3.0`, which is not planned for any time soon, so may still be years away).

Note that use of this function to import data is *OPTIONAL* and in fact *NOT RECOMMENDED*. `respR` has an extremely simple input data structure requirement: paired numeric values of time and oxygen amount in any common units in a `data.frame`. To our knowledge, every oxygen sensor system allows you to export data in a format (e.g. `.csv`, `.txt`, `.xlsx`) which are easy to import into `R`, and this is a basic skill anyone using `R` should be comfortable with. `import_file` was only ever a convenience function intended for those completely new to `R`. It is *ALWAYS* better to import files yourself using generic functions such as `read.csv()`, `read.table()` or `fread()` as it gives you much more control and the ability to troubleshoot issues.

For files currently supported, the function extracts data columns, removes redundant rows of meta-data, and generally cleans up column names (e.g. removes whitespace and characters which cause text encoding issues) to make the data easier to work with. Files should be sensor system raw output files where possible; files opened and re-saved in a different format will likely fail to import.

Currently tested and working for these files:

- Pyro Firesting
- Pyro Workbench (very experimental - likely to fail and will not be updated further)
- PreSens OXY10
- PreSens OXY4
- PreSens (OxyView generic, including multiplate systems)
- PreSens/Loligo 24-Well Multiplate System (output Excel files)
- MiniDOT
- Loligo AutoResp ('_raw' files output, *not* metadata files)
- Loligo Witrox (same as AutoResp, without metadata)
- Vernier (raw qmbl, csv, or txt)
- NeoFox
- Qbox Aqua

Files with European numeric formatting (i.e. commas instead of points to denote decimals) are supported, and will be converted to point decimals on import. This is experimental functionality, so please provide feedback for any files for which this might fail.

While the devices listed above are supported, the import functionality is experimental due to limited access to sample files. Users should be aware we have not been able to test every variation of file formats, and should carefully check the imported data, and be prepared to import data using other functions such as `read.csv()`.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at `https://januarharianto.github.io/respR/`

Value

A `data.frame` object of all columned data

Examples

```
## Not run:
# Import a file
import_file("path/to/file")

# Import a file and export it to same directory as a csv
import_file("path/to/file", export = TRUE)

## End(Not run)
```

inspect *Explore and visualise respirometry data and check for common errors*

Description

`inspect()` is a data exploration and preparation function that visualises respirometry data and checks it for errors that may affect the use of further functions in `respR`. It also subsets specified columns into a new `list` object that can be used in subsequent functions, reducing the need for additional inputs. Note, use of `inspect` to prepare data for the subsequent functions is optional. Functions in `respR` can accept regular `R` data objects including data frames, data tables, tibbles, vectors, etc. It is a quality control and exploratory step to help users view and prepare their data prior to analysis.

Usage

```
inspect(
  x,
  time = NULL,
  oxygen = NULL,
  width = 0.1,
  plot = TRUE,
  add.data = NULL,
  ...
)
```

Arguments

`x`	data.frame. Any object of class `data.frame` (incl. `data.table`, `tibble`, etc.). Should contain paired numeric values of time and oxygen.
`time`	integer or string. Defaults to `1`. Specifies the column of the Time data as either a column number or the name.
`oxygen`	integer or string, or vector of either. Defaults to `2`. Specifies the column(s) of the Oxygen data as either a vector of column numbers or names.
`width`	numeric, 0.01 to 1. Defaults to `0.1`. Width used in the rolling regression plot as proportion of total length of data.
`plot`	logical. Defaults to `TRUE`. Plots the data. If `time` and single `oxygen` columns selected, plots timeseries data, plus plot of rolling rate. If multiple `oxygen` columns, plots all timeseries data only.
`add.data`	integer or string. Defaults to `NULL`. Specifies the column number or name of an optional additional data source that will be plotted in blue alongside the full oxygen timeseries.
`...`	Allows additional plotting controls to be passed, such as `legend = FALSE`, `quiet = TRUE`, `rate.rev = FALSE` and `pos`. A different `width` can also be passed in `plot()` commands on output objects.

Details

Given an input data frame, `x`, the function scans the specified `time` and `oxygen` columns for the following issues. Columns are specified by using the column number (e.g. `time = 1`), or by name (e.g. `time = "Time.Hrs"`). If `time` and `oxygen` are left `NULL` the default of `time = 1, oxygen = 2` is applied.

Check for numeric data:

`respR` requires data be in the form of paired values of numeric time and oxygen. All columns are checked that they contain numeric data before any other checks are performed. If any of the inspected columns do not contain numeric data the remaining checks for that column are skipped, and the function exits returning `NULL`, printing the summary of the checks. No plot is produced. Only when all inspected columns pass this numeric check can the resulting output object be saved and passed to other `respR` functions.

Other checks:

The `time` column is checked for missing (`NA/NaN`) values, positive and negative infinite values (`Inf/-Inf`), that values are sequential, that there are no duplicate times, and that it is numerically evenly-spaced. Oxygen columns are checked for missing (`NA/NaN`) and infinite values (`Inf/-Inf`). See **Failed Checks** section for what it means for analyses if these checks result in warnings. If the output is assigned, the specified `time` and `oxygen` columns are extracted and saved to a `list` object for use in later functions such as `calc_rate()` and `auto_rate()`. A plot is also produced.

Plot:

If `plot = TRUE` (the default), a plot of the oxygen timeseries is produced in the upper panel. In addition, a rolling regression plot in the lower panel shows the rate of change in oxygen across a rolling window specified using the `width` operator (default is `width = 0.1`, or 10% of the entire dataset). This plot provides a quick visual inspection of how the rate varies over the course of the experiment. Regions of stable and consistent rates can be identified on this plot as flat or level areas. This plot is for exploratory purposes only; later functions allow rate to be calculated over specific regions. Each individual rate value is plotted against the centre of the time window used to calculate it.

Note: Since `respR` is primarily used to examine oxygen consumption, the oxygen rate plot is by default plotted on a reverse y-axis. In `respR` oxygen uptake rates are negative since they represent a negative slope of oxygen against time. In these plots the axis is reversed so that higher uptake rates (i.e. more negative) will be higher on these plots. If you are interested instead in oxygen production rates, which are positive, the `rate.rev = FALSE` input can be passed in either the `inspect` call, or when using `plot()` on the output object. In this case, the rate values will be plotted numerically, and higher oxygen *production* rates will be higher on the plot.

Plot an additional data source:

Using the `add.data` input an additional data source, for example temperature, can be plotted alongside the oxygen timeseries. This should be either a column number (e.g. `add.data = 3`) or name (e.g. `add.data = "Temperature"`) indicating a column in the input `x` data frame sharing the same time data. None of the data checks are performed on this column; it is simply to give a basic visual aid in the plot to, for example, help decide if regions of the data should be used or not used because this parameter was variable. Values are saved in the output as a vector under `$add.data`. It is plotted in blue on a separate y-axis on the main timeseries plot. It is *not* plotted if multiple oxygen columns are inspected. See examples.

Additional plotting options:

A different `width` value can be passed to see how it affects estimation of the rolling rate. If axis labels obscure parts of the plot they can be suppressed using `legend = FALSE`. Suppress console output messages with `quiet = TRUE`. If multiple columns have been inspected, the `pos` input can be used to examine each time~oxygen dataset. If axis labels (particularly y-axis) are difficult to read, `las = 2` can be passed to make axis labels horizontal, and `oma` (outer margins, default `oma = c(0.4, 1, 1.5, 0.4)`) or `mai` (inner margins, default `mai = c(0.3, 0.15, 0.35, 0.15)`) can be used to adjust plot margins. See examples.

Multiple Columns of Oxygen Data:

For a quick overview of larger datasets, multiple oxygen columns can be inspected for errors and plotted by using the `oxygen` input to select multiple columns. These must share the same `time` column. In this case, data checks are performed, with a plot of each oxygen time series, but no rolling rate plot is produced. All data are plotted on the same axis range of both time and oxygen (total range of data). This is chiefly exploratory functionality to allow for a quick overview of a dataset, and it should be noted that while the output `inspect` object will contain all columns in its `$dataframe` element, subsequent functions in `respR` (`calc_rate`, `auto_rate`, etc.) will by default only use the first two columns (`time`, and the first specified `oxygen` column). To analyse multiple columns and determine rates, best practice is to inspect and assign each time-oxygen column pair as separate `inspect` objects. See Examples.

Flowthrough Respirometry Data:

For flowthrough respirometry data, see the specialised `inspect.ft()` function.

Failed Checks:

The most important data check in `inspect` is that all data columns are numeric. If any column fails this check, the function skips the remaining checks for that column, the function exits returning `NULL`, and no output object or plot is produced.

The other failed check that requires action is the check for infinite values (`Inf/-Inf`). Some oxygen sensing systems add these in error when interference or data dropouts occur. Infinite values will cause problems when it comes to calculating rates, so need to be removed. If found, locations of these are printed and can be found in the output object under `$locs`. Note, these values are not plotted, so special note should be taken of the warnings and console printout.

The remaining data checks in `inspect` are mainly exploratory and help diagnose and flag potential issues with the data that might affect rate calculations. For instance, long experiments may have had sensor dropouts the user is unaware of. Some might not be major issues. For instance, an uneven time warning can result from using decimalised minutes, which is a completely valid time metric, but happens to be numerically unevenly spaced. As an additional check, if uneven time is found, the minimum and maximum intervals in the time data are in the console output, so a user can see immediately if there are large gaps in the data.

If some of these checks produce warnings, it should *generally* not hinder analysis of the data. `respR` has been coded to rely on linear regressions on exact data values, and not make assumptions about data spacing or order. Therefore issues such as missing or NA/NaN values, duplicate or non-sequential time values, or uneven time spacing should not cause any erroneous rate results, as long as they do not occur over large regions of the data. `inspect` however outputs locations (row numbers) of where these issues occur (located in the `$locs` element of the output), allowing users to amend them before analysis. We would strongly recommend that to be completely confident in any results from analysis of such data, and avoid obscure errors, these issues be addressed before proceeding.

S3 Generic Functions:

Saved output objects can be used in the generic S3 functions `plot()`, `print()` and `summary()`.

- `plot()`: plots the result.
- `print()`: prints a summary of the checks performed on the data. If issues are found, locations (row numbers) are printed (up to first 20 occurrences).
- `summary()`: simple wrapper for `print()` function. See above.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at `https://januarharianto.github.io/respR/`

Value

Output is a `list` object of class `inspect`, with a `$dataframe` containing the specified `time` and oxygen columns, inputs, and metadata which can be passed to `calc_rate()` or `auto_rate()` to determine rates. If there are failed checks or warnings, the row locations of the potentially problematic data can be found in `$locs`.

Examples

```
## By default, assumes time is col 1 and oxygen col2:
inspect(sardine.rd)

## Instead, specify time and oxygen columns as either number or name
inspect(sardine.rd, time = 1, oxygen = 2)
inspect(urchins.rd, time = "time.min", oxygen = "a")

## Use add.data input to plot an additional data type
## (this column is not checked)
inspect(sardine.rd, time = 1, oxygen = 2, add.data = 3)

## Adjust the width of the rolling rate plot:
inspect(sardine.rd, 1, 2, width = 0.2)

## Inspect specific columns in multicolumn datasets:
inspect(urchins.rd, time = 1, oxygen = 4)

## Inspect multiple columns for a quick overview
## of a large dataset:
inspect(urchins.rd, time = 1, oxygen = c(11:19))

## Inspect oxygen production data, use a width that gives
## a better rolling rate, and use extra plotting options to
## suppress legend, and ensure rates are plotted not reversed:
inspect(algae.rd, time = 1, oxygen = 2, width = 0.4,
        legend = FALSE, rate.rev = FALSE)

## Pass additional plotting inputs to override defaults and
## allow better y-axis label visibility
inspect(sardine.rd, time = 1, oxygen = 2,
        las = 1, mai = c(0.3, 0.35, 0.35, 0.15))
```

inspect.ft — *Explore and visualise flowthrough respirometry data and check for errors*

Description

`inspect.ft` is a data exploration and preparation function that visualises flowthrough respirometry data, checks it for common issues, and prepares it for use in later functions in `respR`, such as `calc_rate.ft()`.

Usage

```
inspect.ft(
  x,
  time = NULL,
  out.oxy = NULL,
  in.oxy = NULL,
  in.oxy.value = NULL,
  delta.oxy = NULL,
  plot = TRUE,
  add.data = NULL,
  ...
)
```

Arguments

`x`	`data.frame` containing columns of `time` and `out.oxy` or `delta.oxy` concentrations, and optionally `in.oxy`.
`time`	integer or string. Defaults to `1`. Specifies the column of the Time data as either a column number or the name.
`out.oxy`	integer(s) or string(s). Defaults to `NULL`. Specifies the column(s) of the outflow oxygen data as either column numbers or names.
`in.oxy`	integer(s) or string(s). Defaults to `NULL`. Specifies the column(s) of the inflow oxygen data as either column numbers or names.
`in.oxy.value`	numeric value. Defaults to `NULL`. If there is no continuous `in.oxy` data, this specifies a fixed value of oxygen concentration for inflowing water in same units as `out.oxy`, and is used with `out.oxy` to calculate a `delta.oxy`.
`delta.oxy`	integer(s) or string(s). Defaults to all non-time columns if no other inputs given. Specifies the column number(s) or name(s) of delta oxygen data, for when the user has already calculated the difference between outflow and inflow oxygen (should be negative values for oxygen uptake). If this is used, `out.oxy` and `in.oxy` should be NULL.
`plot`	logical. Defaults to TRUE. Plots the data. See Details.

`add.data`	integer or string. Defaults to `NULL`. Specifies the column number or name of an optional additional data source that will be plotted in blue alongside the full oxygen timeseries.
`...`	Allows additional plotting controls to be passed, such as `legend = FALSE`, `quiet = TRUE`, `rate.rev = FALSE` and `pos`.

Details

`inspect.ft` is intended to be specific to *flowthrough* respirometry data. In flowthrough respirometry (also known as 'open flow' or 'continuous flow' respirometry) rather than calculating a rate from a changing oxygen concentration recording in a sealed chamber, instead the difference (i.e. 'oxygen delta') between the inflowing and outflowing oxygen concentrations of a respirometer receiving water at a constant flow rate is used to calculate an oxygen consumption or production rate, typically after it has reached a steady state. Therefore, in general, regions of stable oxygen delta values (difference between outflow and inflow oxygen) are of interest. `inspect.ft` visualises and prepares the data for use in `calc_rate.ft()`. By specifying data types in this function and saving the output, they do not need to be specified in later functions.

Inputs:

Given an input data frame `x`, the function scans the columns specified via the `time`, `out.oxy`, `in.oxy` or `delta.oxy` inputs. Columns are specified by using the column number (e.g. `time = 1`), or by name (e.g. `time = "Time.Hrs"`). If no columns are specified, by default the function assumes the first column is `time`, and all others are `delta.oxy` oxygen data. However, best practice is to use the inputs to specify particular columns.

The `x` input must contain at least two data types; a single column of numeric `time` data, with *either* a column of paired `out.oxy` concentrations (i.e. the exhalent or 'downstream' concentrations), *or* a column of already calculated `delta.oxy` values, that is the difference between outflow and inflow concentrations, or the outflow concentration corrected by a background recording from a 'blank' or empty chamber.

out.oxy input option: If an `out.oxy` column has been specified, in order to calculate the oxygen delta (and therefore a rate in `calc_rate.ft()`) there must also be an inflow oxygen concentration input (i.e. the inhalent or 'upstream' concentration). This will generally be a column of paired `in.oxy` concentrations, in which case the paired values of `out.oxy` and `in.oxy` are used to calculate the oxygen `delta.oxy`, which is saved in the output and used to determine a rate in `calc_rate.ft()`. Alternatively, if the inflow oxygen concentration is a known, generally unvarying value (such as fully air-saturated water from a header tank) this can be entered as a single value via `in.oxy.value` and this is used to calculate the `delta.oxy`.

delta.oxy input option: If delta oxygen values have already been calculated, these can be entered via the `delta.oxy` input, and these are prepared and saved for rate calculations in `calc_rate.ft`.

Check for numeric data:

`respR` requires data be in the form of paired values of numeric time and oxygen. All columns are checked that they contain numeric data before any other checks are performed. If any of the inspected columns do not contain numeric data the remaining checks for that column are skipped, and the function exits returning `NULL`, printing the summary of the checks. No plot is produced. Only when all inspected columns pass this numeric check can the resulting output object be saved and passed to other `respR` functions.

Other checks:

The `time` column is checked for missing (`NA/NaN`) values, infinite values both positive and negative (`Inf/-Inf`), that values are sequential, that there are no duplicate times, and that it is numerically evenly-spaced. Oxygen columns are checked for missing (`NA/NaN`) and infinite values (`Inf/-Inf`). See **Failed Checks** section for what it means for analyses if these checks result in warnings. If the output is assigned, the specified columns are saved to a `list` object for use in later functions such as `calc_rate.ft()`. A plot is also produced.

Plot:

If `plot = TRUE`, entered data is plotted against both time (bottom, blue axis) and row index (top, red axis), depending on the inputs:

- a single `out.oxy` column with either a paired `in.oxy` column or `in.oxy.value`: a two panel plot. The top plot is both outflow (green points) and inflow (turquoise points) oxygen. The bottom plot is the oxygen delta (black points) between outflow and inflow oxygen, essentially a unitless oxygen uptake or production rate.
- a single `delta.oxy` column: a one panel plot of oxygen delta values.
- multiple `out.oxy` or `delta.oxy` columns: a grid plot of all `delta.oxy` data (either as entered or calculated from `out.oxy` and `in.oxy`). Specific delta plots can be examined individually by using the `pos` input (e.g. `plot(x, pos = 2)`). Y-axes are not equal.
- unspecified columns: all columns are plotted assuming `time` is in column 1, and all others are oxygen `delta.oxy` data. Y-axes are not equal.

In delta plots, that is those plotting `delta.oxy` values, either directly entered or calculated, consistent oxygen uptake or production rates will be represented by flat or level regions. The `width` input may help with selecting regions from which to extract rates, and can be passed in the main function call or using `plot()` on the output object. This smooths delta oxygen values by calculating a rolling mean across the data. See **Additional plotting options** below.

Note: Since `respR` is primarily used to examine oxygen consumption, the delta oxygen and rate plots are by default plotted on a reverse y-axis. In `respR` oxygen uptake rates are negative since they represent a negative slope of oxygen against time. In these plots the axis is reversed so that higher uptake rates (i.e. more negative) will be higher on these plots. If you are interested instead in oxygen production rates, which are positive, the `rate.rev = FALSE` input can be passed in either the `inspect.ft` call, or when using `plot()` on the output object. In this case, the delta and rate values will be plotted numerically, and higher oxygen *production* rates will be higher on the plot.

Plot an additional data source:

Using the `add.data` input an additional data source, for example temperature, can be plotted alongside the oxygen timeseries. This should be either a column number (e.g. `add.data = 3`) or name (e.g. `add.data = "Temperature"`) indicating a column in the input `x` data frame sharing the same time data. None of the data checks are performed on this column; it is simply to give a basic visual aid in the plot to, for example, help decide if regions of the data should be used or not used because this parameter was variable. Values are saved in the output as a vector under `$add.data`. It is plotted in blue on a separate y-axis on the main timeseries plot. It is *not* plotted if multiple oxygen columns are inspected. See examples.

Additional plotting options:

The `width` input may help with selecting regions from which to extract rates. This smooths delta oxygen values by calculating a rolling mean across the data, and should be a value between 0 and

1 representing a proportion of the total data width. If left as the default NULL no smoothing is performed. This is a visual aid which only affects plotted values and does not alter output delta oxygen values.

If the legend or labels obscure part of the plot, they can be suppressed via legend = FALSE in either the inspect.ft call, or when using plot() on the output object. Suppress console output messages with quiet = TRUE. If multiple columns have been inspected, the pos input can be used to examine each out.oxy~in.oxy~del.oxy dataset. If axis labels (particularly y-axis) are difficult to read, las = 2 can be passed to make axis labels horizontal. In addition, oma (outer margins, default oma = c(0.4, 1, 1.5, 0.4)), and mai (inner margins, default mai = c(0.3, 0.15, 0.35, 0.15)) can be used to adjust plot margins.

Multiple data columns:

For a quick overview of larger experiments, multiple columns of out.oxy, in.oxy and delta.oxy can be inspected, but must share the same numeric time data column specified by the time input. Note, multiple column inspection is chiefly intended to be exploratory functionality to provide a quick overview of larger datasets. While the output will contain all data columns in $dataframe and $data, subsequent functions such as calc_rate.ft() will use only the first delta.oxy column for calculating rates. Best practice is to inspect and assign each individual experiment or column pair as separate inspect.ft objects. See Examples.

If multiple out.oxy columns are specified, in.oxy can be a single column (if for example all chambers are supplied from the same header tank), in which case it is used to calculate an oxygen delta for all out.oxy columns. A single in.oxy.value in the same units as out.oxy can also be specified. There can also be multiple in.oxy columns, in which case it is assumed each out.oxy column is paired with each in.oxy at the same position, and used to calculate the oxygen delta.oxy. In this case, out.oxy and in.oxy must have equal numbers of columns.

Failed Checks:

The most important data check in inspect.ft is that all data columns are numeric. If any column fails this check, the function skips the remaining checks for that column, the function exits returning NULL, and no output object or plot is produced.

The other failed check that requires action is the check for infinite values (Inf/-Inf). Some oxygen sensing systems add these in error when interference or data dropouts occur. Infinite values will cause problems when it comes to calculating rates, so need to be removed. If found, locations of these are printed and can be found in the output object under $locs. Note, these values are not plotted, so special note should be taken of the warnings and console printout.

The remaining data checks in inspect.ft are mainly exploratory and help diagnose and flag potential issues with the data that might affect rate calculations. For instance, long experiments may have had sensor dropouts the user is unaware of. Some might not be major issues. For instance, an uneven time warning can result from using decimalised minutes, which is a completely valid time metric, but happens to be numerically unevenly spaced. As an additional check, if uneven time is found, the minimum and maximum intervals in the time data are in the console output, so a user can see immediately if there are large gaps in the data.

If some of these checks produce warnings, it should *generally* not hinder analysis of the data. respR has been coded to rely on linear regressions on exact data values, and not make assumptions about data spacing or order. Therefore issues such as missing or NA/NaN values, duplicate or non-sequential time values, or uneven time spacing should not cause any erroneous results, as long as they do not occur over large regions of the data. inspect.ft however outputs locations (row numbers) of where these issues occur (located in the $locs element of the output), allowing users to amend them before analysis. We would recommend that to be completely confident in any results from analysis of such data, and avoid obscure errors, these issues be addressed before proceeding.

Background control or "blank" experiments:

For experiments in which the specimen data is to be background corrected by a concurrently-run control experiment, `inspect.ft` can be used by specifying the specimen experiment as `out.oxy`, and the "blank" as the `in.oxy` input. In this way, any variations in oxygen in the specimen data due to background microbial activity, or for any other reason such as fluctuations in inflow oxygen, are accounted for in the delta oxygen calculations, and therefore in the rate calculated in `calc_rate.ft()`. See the vignettes on the website for examples.

If the background recordings are experiments with their own outflow and inflow recordings, which show a generally consistent oxygen delta due to microbial activity, this can be saved as a separate `inspect.ft` object, a background rate calculated in `calc_rate.ft()`, and this used in `adjust_rate.ft()` as the `by` input to perform background adjustments to specimen rates.

Note: All background calculations should be from experiments done at the *same flow rate* as the specimen experiments to be corrected.

S3 Generic Functions:

Saved output objects can be used in the generic S3 functions `plot()`, `print()` and `summary()`.

- `plot()`: plots the result.
- `print()`: prints a summary of the checks performed on the data. If issues are found, locations (row numbers) are printed (up to first 20 occurrences).
- `summary()`: simple wrapper for `print()` function. See above.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at https://januarharianto.github.io/respR/

Value

Output is a `list` object of class `inspect.ft` containing input parameters and data, data check summaries, and metadata, which can be passed to `calc_rate.ft()` to determine rates. If there are failed checks or warnings, the row locations of the potentially problematic data can be found in `$locs`.

Examples

```
# Inspect outflow and inflow oxygen data
x <- inspect.ft(flowthrough.rd, time = 1, out.oxy = 2,
                in.oxy = 3)
print(x)
plot(x)

# Inspect outflow oxygen data with inflow oxygen as a known value in
# the same units
x <- inspect.ft(flowthrough.rd, time = 1, out.oxy = 2,
                in.oxy.value = 8.90)
```

```
# Inspect already calculated delta oxygen data
inspect.ft(flowthrough.rd, time = 1, delta.oxy = 4)

# inspect multiple columns for a quick overview
inspect.ft(flowthrough_mult.rd, time = 1, delta.oxy = 10:12)

# Inspect outflow and use a blank control chamber as background
# correction
#
# This experiment has increasing background respiration over time.
# Inspecting outflow oxygen with inflow header tank concentrations
# suggests specimen rates (bottom delta.oxy plot) are increasing.
inspect.ft(flowthrough_sim.rd, time = 1,
           out.oxy = 2, in.oxy = 4)

# However, inspecting with recordings from a concurrent blank
# control accounts for this and shows specimen rates are level
# when background is taken into account.
inspect.ft(flowthrough_sim.rd, time = 1,
           out.oxy = 2, in.oxy = 3)
```

`intermittent.rd` *Respirometry data of the sea urchin,* Heliocidaris Erythrogramma

Description

Multiple measurements of oxygen consumption in a single sea urchin, *Heliocidaris erythrogramma*, obtained using intermittent flow respirometry. The experiment was conducted at the Sydney Institute of Marine Science in Sydney, Australia. There are a total of 3 replicates showing declining oxygen, separated by flushes where new water was added showing increasing oxygen. Data was collected using a Vernier Optical DO probe (ODO-BTA).

Usage

```
intermittent.rd
```

Format

A data frame object consisting of 2 columns (time and dissolved oxygen) and 4831 rows (approx 80 min of data).

Details

- Dissolved oxygen units: `mg/L`
- Time units: `seconds`
- Chamber volume (L): `2.379`
- Specimen ash-free dry mass (kg): `0.006955`

Replicate structure (Rows - Experiment section):

- `1:1900` - Replicate 1
- `1901:2100` - Flush 1
- `2101:3550` - Replicate 2
- `3551:3900` - Flush 2
- `3901:4831` - Replicate 3

Author(s)

Nicholas Carey

`oxy_crit` *Calculate critical oxygen values, such as PCrit*

Description

Identifies critical oxygen values, the oxygen tension or concentration below which an uptake rate transitions from independent to dependent on the oxygen supply, typically known as *PCrit*.

Usage

```
oxy_crit(
  x,
  method = "bsr",
  time = NULL,
  oxygen = NULL,
  rate = NULL,
  width = 0.1,
  parallel = FALSE,
  thin = 5000,
  plot = TRUE,
  ...
)
```

Arguments

`x`	object of class `inspect` or a `data.frame` containing either paired oxygen~time values, or paired rate~oxygen values. See Details.
`method`	string. Defaults to `"bsr"`. Critical oxygen value analysis method. Either `"bsr"` or `"segmented"`. See Details.
`time`	integer or string. Defaults to 1. Specifies column number or column name of the time data.
`oxygen`	integer or string. Defaults to 2. Specifies column number or column name of the oxygen data.
`rate`	integer or string. Defaults to NULL. Specifies column number or column name of the rate data.

`width`	numeric value between 0 and 1 representing proportion of the total data length. Determines the width of the rolling regression used to determine the rolling rate and the rolling mean of oxygen values the rate is paired with. Defaults to 0.1, representing 10% of total rows.
`parallel`	logical. Defaults to FALSE. Enables parallel processing for computationally intensive analyses of large datasets.
`thin`	integer. Defaults to 5000. Number of rows to subsample `x` data to before running `"bsr"` analysis. No effect on datasets smaller than this value or with `"segmented"` method. To perform no subsampling enter as `NULL`. See Details.
`plot`	logical. Defaults to TRUE.
`...`	Allows additional plotting controls to be passed, such as `legend = FALSE`, `quiet = TRUE`, `rate.rev = FALSE`, and `panel`. See Plotting section.

Details

In earlier versions of `respR`, this function was known as `pcrit` or `calc_pcrit`. It was renamed to avoid conflicts with functions of the same name in another package, and also because technically the *P* in *PCrit* stands for the partial *pressure* of oxygen. Since the function returns the value in the units of the data as entered, whether they are concentration or pressure units, this terminology can be technically in error. Instead, for the purposes of the documentation we refer to this as the *Critical Oxygen Value*, or "*COV*". If the units of oxygen are partial pressure units (e.g. kPa), this is equivalent to PCrit, otherwise they should be reported with this in mind.

Methods:

The `oxy_crit()` function provides two methods (one of which outputs two results) to calculate the *COV*. These are selected using the `method` input.

Broken Stick Regression: `method = "bsr"`*:*
This is the default method, adapted from the “Broken-Stick” regression (*BSR*) approach, of Yeager & Ultsch (1989), in which two segments of the data are iteratively fitted and the intersection with the smallest sum of the residual sum of squares between the two linear models is the estimated *COV*. Two slightly different ways of reporting this breakpoint are detailed by Yeager & Ultsch (1989); the *intercept* and *midpoint*. These are usually very close in value, and the function returns both.
The `thin` input influences the *BSR* analysis. The method is very computationally intensive, so to speed up analyses the `thin` input will subsample datasets longer than this input to this number or rows before analysis. The default value of 5000 has in testing provided a good balance between speed and results accuracy and repeatability. However, results may vary with different datasets, so users should experiment with varying the value. To perform no subsampling and use the entire dataset enter `thin = NULL`. It has no effect on datasets shorter than the `thin` input.

Segmented Regression: `method = "segmented"`*:*
The second method is a wrapper for the "Segmented" regression approach, available as part of the `segmented` R package (Muggeo 2008), which estimates the *COV* by iteratively fitting two intersecting models and selecting the value that minimises the “gap” between the fitted lines.

Inputs:

The data input `x` should be an `inspect` object or `data.frame` containing oxygen~time data, or a `data.frame` containing rate~oxygen data.

Oxygen ~ Time data:
This is the typical input, where a timeseries of oxygen concentrations or partial pressures against time has been recorded, generally down to a very low value of oxygen. A column of `time` and a column of `oxygen` should be specified. The function defaults to `time = 1` and `oxygen = 2` if no other inputs are entered. These can also be specified using the column names.
If an `inspect` object is entered as the `x` input, the data frame is extracted automatically and column identifiers are not required since these were already entered in `inspect`. Note, if multiple `oxygen` columns were entered in `inspect` only the first entered one will be used in `oxy_crit`.
To calculate the *COV*, the function requires data in the form of oxygen uptake rate against oxygen value. Therefore, the function performs a rolling regression on the oxygen~time data to determine rates, and pairs these against a rolling mean of the oxygen data. The function then performs the selected analysis `method` on these data. The width of the rolling regression and rolling mean is determined by the `width` input. The default is 0.1, representing 10% of the length of the data. This performs well in testing, however performance may vary with data that has abrupt changes in rate, or is particularly noisy. Users should experiment with different `width` values to see how it affects results, and report this with their results and analysis parameters.

Rate ~ Oxygen data:
Alternatively, if existing rolling oxygen uptake rates have been calculated, and have appropriate paired oxygen concentration or partial pressure values, these can be entered with the `rate` and `oxygen` inputs specifying the respective columns as either numbers or the column names. In this case the function performs the selected analysis `method` on these data directly without any processing. The `width` input in this case is not relevant and is ignored.
This option can only be used with data frame `x` inputs. Note, other columns such as time data may be present in the input, but are not required so need not be specified.

Plot:

A plot is produced (provided `plot = TRUE`) of the input data and results. The top panel is the input data, either the oxygen~time timeseries, or the rate~oxygen series, depending on what was entered in `x`. If the former, the critical oxygen value is indicated by a horizontal line, or two lines in the case of the Broken-Stick analysis. Note, since the two *BSR* results are usually close in value these may overlay each other.

The bottom plot is the rate~oxygen series upon which the analysis was conducted, either as input or as calculated. Critical oxygen values are indicated by vertical lines, and regression fits upon which the analysis was based by black dashed lines.

Note, that in `respR` oxygen uptake rates are negative since they represent a negative slope of oxygen against time, therefore by default rates are plotted on a reverse y-axis so higher rates appear higher on the plot. If analysing already calculated rates which are positive values this behaviour can be reversed by passing `rate.rev = FALSE` in either the main function call or when calling `plot()` on the output object. There is no issue with using positive rate values; they will give identical critical value results in the analysis.

Additional plotting options:
If the legend obscures parts of the plot they can be suppressed using `legend = FALSE`. Suppress console output messages with `quiet = TRUE`. Each panel can be plotted on its own using `panel = 1` or `panel = 2`. If using already-calculated, positive rate values to identify critical oxygen values, the y-axis of the rolling rate plot can be plotted *not* reversed by passing `rate.rev = FALSE` These inputs can be passed in either the main `oxy_crit` call or when calling `plot()` on the output object. If axis labels (particularly y-axis) are difficult to read, `las = 2` can be passed

to make axis labels horizontal, and oma (outer margins, default `oma = c(0.4, 1, 1.5, 0.4)`), and `mai` (inner margins, default `mai = c(0.3, 0.15, 0.35, 0.15)`) used to adjust plot margins.

S3 Generic Functions:

Saved output objects can be used in the generic S3 functions `print()` and `summary()`.

- `print()`: prints the critical oxygen value for the particular `method` used.
- `summary()`: prints critical oxygen value, plus additional coefficients and metadata for the particular `method` used. See Yeager & Ultsch (1989) and Muggeo (2008) for what these represent. The summary can be exported as a separate data frame by passing `export = TRUE`.

More:

For additional help, documentation, vignettes, and more visit the `respR` website at `https://januarharianto.github.io/respR/`

Value

Output is a `list` object of class `oxy_crit` containing input parameters and data, various summary data, metadata, and the primary output of interest `$crit`, which is the critical oxygen value in the units of the oxygen data as entered. This can be converted to additional units using `convert_DO()`. Note, if the Broken-Stick analysis (`method == "bsr"`) has been used, `$crit` will contain two results; `$crit.intercept` and `$crit.midpoint`. For full explanation of the difference between these see Yeager & Ultsch (1989), however they are generally very close in value.

References

Yeager DP, Ultsch GR (1989) Physiological regulation and conformation: A BASIC program for the determination of critical points. Physiological Zoology 62:888–907. doi: 10.1086/physzool.62.4.30157935

Muggeo V (2008) Segmented: an R package to fit regression models with broken-line relationships. R News 8:20–25.

Examples

```
## Run on oxygen~time data.frame with default inputs
oxy_crit(squid.rd)

## Try a lower 'thin' input to speed up analysis
oxy_crit(squid.rd, thin = 1000)

## Use the Segmented method instead
oxy_crit(squid.rd, method = "segmented")

## Experiment with different 'width' input
# Higher widths tend to oversmooth data
oxy_crit(squid.rd, method = "segmented", width = 0.2)
# Lower width in this case gives very similar result to default 0.1
oxy_crit(squid.rd, method = "segmented", width = 0.05)

## Run on oxygen~time data in 'inspect' object
insp <- inspect(squid.rd, time = 1, oxygen = 2)
```

```
oxy_crit(insp)

## Run on already calculated rate~oxygen data
# Calculate a rolling rate
rate <- auto_rate(squid.rd,
                  method = "rolling",
                  width = 0.1,
                  plot = FALSE)$rate

## Calculate a rolling mean oxygen
oxy <- na.omit(roll::roll_mean(squid.rd[[2]],
                               width = 0.1 * nrow(squid.rd)))
## Combine to data.frame
squid_rate_oxy <- data.frame(oxy, rate)
## Perform COV analysis
oxy_crit(squid_rate_oxy, oxygen = 1, rate = 2)
```

`sardine.rd` *Respirometry data of the sardine,* Sardinops sagax

Description

A single experiment on the sardine species *Sardinops sagax* in a Loligo Systems swim tunnel and Witrox oxygen probe system. There are three columns: `$Time` in seconds, `$Oxygen` content recorded in percent air saturation, and `$Temperature` in °C. Mean temperature, salinity and atmospheric pressure are supplied below to allow for conversion to oxygen concentration units.

Usage

```
sardine.rd
```

Format

A data frame object consisting of 3 columns (time, % air saturation and temperature) and 7513 rows (approx 2.1h of data).

Details

Experiment conducted at Hopkins Marine Station, Stanford University, Pacific Grove, California.

- Dissolved oxygen units: `% air saturation`
- Time units: `seconds`
- Chamber volume (L): `12.3`
- Specimen wet mass (kg): `0.0477`
- Temperature (°C): `14.8`
- Salinity: `35`
- Atm. Pressure (bar): `1.013253`

Author(s)

Nicholas Carey

`select_rate` *Select rate results based on a range of criteria*

Description

The functions in `respR` are powerful, but outputs can be large and difficult to explore, especially when there are hundreds to thousands of results, for example the output of `auto_rate` on large datasets, or the outputs of `calc_rate.int` from long intermittent-flow experiments.

The `select_rate` and `select_rate.ft` functions help explore, reorder, and filter `convert_rate` and `convert_rate.ft` results according to various criteria. For example, extracting only positive or negative rates, only the highest or lowest rates, only those from certain data regions, and numerous other methods that allow advanced filtering of results so the final selection of rates is well-defined towards the research question of interest. This also allows for highly consistent reporting of results and rate selection criteria.

Multiple selection criteria can be applied by saving the output and processing it through the function multiple times using different methods, or alternatively via piping (`%>%` or `%>%`). See Examples.

Note: when choosing a `method`, keep in mind that to remain mathematically consistent, `respR` outputs oxygen consumption (i.e. respiration) rates as negative values. This is particularly important in the difference between `highest/lowest` and `minimum/maximum` methods. See Details.

When a rate result is omitted by the selection criteria, it is removed from the `$rate.output` element of the `convert_rate` object, and the associated data in `$summary` (i.e. that row) is removed. Some methods can also be used with an `n = NULL` input to reorder the `$rate` and `$summary` elements in various ways.

Replicate and Rank columns:
The summary table `$rank` column is context-specific, and what it represents depends on the type of experiment analysed or the function used to determine the rates. If numeric values were converted, it is the order in which they were entered. Similarly, if `calc_rate` was used, it is the order of rates as entered using `from` and `to` (if multiple rates were determined). For `auto_rate` it relates to the `method` input. For example it indicates the kernel density ranking if the `linear` method was used, the ascending or descending ordering by absolute rate value if `lowest` or `highest` were used, or the numerical order if `minimum` or `maximum` were used. For intermittent-flow experiments analysed via `calc_rate.int` and `auto_rate.int` these will be ranked *within* each replicate as indicated in the `$rep` column. The `$rep` and `$rank` columns can be used to keep track of selection or reordering because the original values will be retained unchanged through selection or reordering operations. The original order can always be restored by using `method = "rep"` or `method = "rank"` with `n = NULL`. In both these cases the `$summary` table and `$rate.output` will be reordered by `$rep` (if used) then `$rank` to restore the original ordering.

Note that if you are analysing intermittent-flow data and used `auto_rate.int` but changed the `n` input to output more than one rate result per replicate, the selection or reordering operations will not take any account of this. You should carefully consider if or why you need to output multiple rates per replicate in the first place. If you have, you can perform selection on individual replicates by using `method = "rep"` to select individual replicates then apply additional selection criteria.

Usage

```
select_rate(x, method = NULL, n = NULL)

select_rate.ft(x, method = NULL, n = NULL)
```

Arguments

`x`	list. An object of class `convert_rate` or `convert_rate_select`.
`method`	string. Method by which to select or reorder rate results. For most methods matching results are *retained* in the output. See Details.
`n`	numeric. Number, percentile, or range of results to retain or omit depending on `method`. Default is `NULL`, in which case some methods will instead reorder the results. See Details.

Details

These are the current methods by which rates in `convert_rate` objects can be selected. Matching results are *retained* in the output. Some methods can also be used to reorder the results. Note that the methods selecting by rate value operate on the `$rate.output` element, that is the final converted rate value.

`positive`, `negative`:

Selects all `positive` (>0) or `negative` (<0) rates. `n` is ignored. Useful, for example, in respirometry on algae where both oxygen consumption and production rates are recorded. Note, `respR` outputs oxygen consumption (i.e. respiration) rates as *negative* values, production rates as *positive*.

`nonzero`, `zero`:

Retains all `nonzero` rates (i.e. removes any zero rates), or retains *only* `zero` rates (i.e. removes all rates with any value). `n` is ignored.

`lowest`, `highest`:

These methods can only be used when rates all have the same sign, that is are all negative or all positive. These select the lowest and highest **absolute** rate values. For example, if rates are all negative, `method = 'highest'` will retain the highest magnitude rates regardless of the sign. `n` should be an integer indicating the number of lowest/highest rates to retain. If `n = NULL` the results will instead be reordered by lowest or highest rate without any removed. See `minimum` and `maximum` options for extracting *numerically* lowest and highest rates.

`lowest_percentile`, `highest_percentile`:

These methods can also only be used when rates all have the same sign. These retain the `n`'th lowest or highest percentile of **absolute** rate values. For example, if rates are all negative `method = 'highest_percentile'` will retain the highest magnitude `n`'th percentile regardless of the sign. `n` should be a percentile value between 0 and 1. For example, to extract the lowest 10th percentile of absolute rate values, you would enter `method = 'lowest_percentile', n = 0.1`.

`minimum`, `maximum`:

In contrast to lowest and highest, these are *strictly numerical* options which take full account of the sign of the rate, and can be used where rates are a mix of positive and negative. For example, method = 'minimum' will retain the minimum numerical value rates, which would actually be the highest oxygen uptake rates. n is an integer indicating how many of the min/max rates to retain. If n = NULL the results will instead be reordered by minimum or maximum rate without any removed.

minimum_percentile, maximum_percentile:

Like min and max these are *strictly numerical* inputs which retain the n'th minimum or maximum percentile of the rates and take full account of the sign. Here n should be a percentile value between 0 and 1. For example, if rates are all negative (i.e. typical uptake rates), to extract the lowest 10th percentile of rates, you would enter method = 'maximum_percentile', n = 0.1. This is because the *lowest* negative rates are numerically the *maximum* rates (highest/lowest percentile methods would be a better option in this case however).

rate:

Allows you to enter a value range of output rates to be retained. Matching regressions in which the rate value falls within the n range (inclusive) are retained. n should be a vector of two values. For example, to retain only rates where the rate value is between 0.05 and 0.08: method = 'rate', n = c(0.05, 0.08). Note this operates on the $rate.output element, that is converted rate values.

rep, rank:

These refer to the respective columns of the $summary table. For these, n should be a numeric vector of integers of rep or rank values to retain. To retain a range use regular R syntax, e.g. n = 1:10. If n = NULL no results will be removed, instead the results will be reordered ascending by rep (if it contains values) then rank. Essentially this restores the original ordering if other reordering operations have been performed.

The values in these columns depend on the functions used to calculate rates. If calc_rate was used, rep is NA and rank is the order of rates as entered using from and to (if multiple rates were determined). For auto_rate, rep is NA and rank relates to the method input. For example it indicates the kernel density ranking if the linear method was used, the ascending or descending ordering by absolute rate value if lowest or highest were used, or by numerical order if minimum or maximum were used. If calc_rate.int or auto_rate.int were used, rep indicates the replicate number and the rank column represents rank *within* the relevant replicate, and will generally be filled with the value 1. Therefore you need to adapt your selection criteria appropriately towards which of these columns is relevant.

rep_omit, rank_omit:

These refer to the rep and rank columns of the $summary table and allow you to exclude rates from particular replicate or rank values. For these, n should be a numeric vector of integers of rep or rank values to OMIT. To omit a range use regular R syntax, e.g. n = 1:10.

rsq, row, time, density:

These methods refer to the respective columns of the $summary data frame. For these, n should be a vector of two values. Matching regressions in which the respective parameter falls within the n range (inclusive) are retained. To retain all rates with a R-Squared 0.90 or above: method = 'rsq', n = c(0.9, 1). The row and time ranges refer to the $row-$endrow or $time-$endtime columns and the original raw data ($dataframe element of the convert_rate object), and can be used to constrain results to rates from particular regions of the data (although usually a better option is to subset_data() prior to analysis). Note time is not the same as duration - see later section - and row refers to rows of the raw data, **not** rows of the summary table - see `manual` method for this. For all of these methods, if `n = NULL` no results will be removed, instead the results will be reordered by that respective column (descending for `rsq` and `density`, ascending for `row`, and `time`).

`intercept`, `slope`:

These methods are similar to the above and refer to the `intercept_b0` and `slope_b1` summary table columns. Note these linear model coefficients represent different things in flowthrough vs. other analyses. In non-flowthrough analyses slopes represent rates and coefficients such as a high r-squared are important. In flowthrough, slopes represent the stability of the data region, in that the closer the slope is to zero, the less the delta oxygen values in that region vary, which is an indication of a region of stable rates. In addition, intercept values close to the calculated mean delta of the region also indicate a region of stable rates. Therefore these methods are chiefly useful in selection of flowthrough results, for example slopes close to zero. If `n = NULL` no results will be removed, instead the results will be reordered by ascending value by that column.

`time_omit`, `row_omit`:

These methods refer to the original data, and are intended to *exclude* rates determined over particular data regions. This is useful in the case of, for example, a data anomaly such as a spike or sensor dropout. For these inputs, `n` are values (a single value, multiple values, or a range) indicating data timepoints or rows of the original data to exclude. Only rates (i.e. regressions) which *do not* utilise those particular values are retained in the output. For example, if an anomaly occurs precisely at timepoint 3000, `time_omit = 3000` means only rates determined solely over regions before or after this will be retained. If it occurs over a range this can be entered as, `time_omit = c(3000,3200)`. If you want to exclude a regular occurrence, for example the flushes in intermittent-flow respirometry, or any other non-continuous values they can be entered as a vector, e.g. `row_omit = c(1000, 2000, 3000)`. Note this last option can be extremely computationally intensive when the vector or dataset is large, so should only be used when a range cannot be entered as two values, which is much faster. For both methods, input values must match exactly to values present in the dataset.

`oxygen`:

This can be used to constrain rate results to regions of the data based on oxygen values. `n` should be a vector of two values in the units of oxygen in the raw data. Only rate regressions in which all datapoints occur within this range (inclusive) are retained. Any which use even a single value outside of this range are excluded. Note the summary table columns `oxy` and `endoxy` refer to the first and last oxygen values in the rate regression, which should broadly indicate which results will be removed or retained, but this method examines *every* oxygen value in the regression, not just first and last.

`oxygen_omit`:

Similar to `time_omit` and `row_omit` above, this can be used to *omit* rate regressions which use particular oxygen values. For this `n` are values (single or multiple) indicating oxygen values in the original raw data to exclude. Every oxygen value used by each regression is checked, and to be excluded an `n` value must match *exactly* to one in the data. Therefore, note that if a regression is fit across the data region where that value would occur, it is not necessarily excluded unless that *exact value* occurs. You need to consider the precision of the data values recorded. For example, if you wanted to exclude any rate using an oxygen value of `7`, but your data are recorded to two decimals, a rate fit across these data would *not* be excluded: `c(7.03, 7.02, 7.01, 6.99, 6.98, ...)`. To get around this you can use regular R syntax to input vectors at the correct precision, such as seq, e.g. `seq(from = 7.05, to = 6.96, by = -0.01)`. This can be used to input ranges of oxygen values to exclude.

`duration`:

This method allows selection of rates which have a specific duration range. Here, `n` should be a numeric vector of two values. Use this to set minimum and maximum durations in the time units of the original data. For example, `n = c(0,500)` will retain only rates determined over a maximum of 500 time units. To retain rates over a minimum duration, set this using the minimum value plus the maximum duration or simply infinity. For example, for rates determined over a minimum of 500 time units `n = c(500,Inf))`

`manual`:

This method simply allows particular rows of the `$summary` data frame to be manually selected to be retained. For example, to keep only the top row `method = 'manual', n = 1`. To keep multiple rows use regular `R` selection syntax: `n = 1:3`, `n = c(1,2,3)`, `n = c(5,8,10)`, etc. No value of `n` should exceed the number of rows in the `$summary` data frame. Note this is not necessarily the same as selecting by the `rep` or `rank` methods, as the table could already have undergone selection or reordering.

`manual_omit`:

As above, but this allows particular rows of the `$summary` data frame to be manually selected to be *omitted*.

`overlap`:

This method removes rates which overlap, that is regressions which are partly or completely fit over the same rows of the original data. This is useful in particular with `auto_rate` results. The `auto_rate linear` method may identify multiple linear regions, some of which may substantially overlap, or even be completely contained within others. In such cases summary operations such as taking an average of the rate values may be questionable, as certain values will be weighted higher due to these multiple, overlapping results. This method removes overlapping rates, using `n` as a threshold to determine degree of permitted overlap. It is recommended this method be used after all other selection criteria have been applied, as it is quite aggressive about removing rates, and can be *very* computationally intensive when there are many results.

While it can be used with `auto_rate` results determined via the `rolling`, `lowest`, or `highest` methods, by their nature these methods produce *all possible* overlapping regressions, ordered in various ways, so other selection methods are more appropriate. The `overlap` method is generally intended to be used in combination with the `auto_rate linear` results, but may prove useful in other analyses.

Permitted overlap is determined by `n`, which indicates the proportion of each particular regression which must overlap with another for it to be regarded as overlapping. For example, `n = 0.2` means a regression would have to overlap with at least one other by at least 20% of its total length to be regarded as overlapping.

The `"overlap"` method performs two operations:

First, regardless of the `n` value, any rate regressions which are completely contained within another are removed. This is also the only operation if `n = 1`.

Secondly, for each regression in `$summary` starting from the bottom of the summary table (usually the lowest ranked result, but this depends on the analysis used and if any reordering has been already occurred), the function checks if it overlaps with any others (accounting for `n`). If not, the next lowest is checked, and the function progresses up the summary table until it finds one that does. The first to be found overlapping is then removed, and the process repeats starting again from the bottom of the summary table. If no reordering to the results has occurred, this means lower ranked results are removed first. This is repeated iteratively until only non-overlapping rates (accounting for `n`) remain.

If `n = 0`, only rates which do not overlap at all, that is share *no* data, are retained. If `n = 1`, only rates which are 100% contained within at least one other are removed.

Reordering results:

Several methods can be used to reorder results rather than select them, by not entering an `n` input (that is, letting the `n = NULL` default be applied). Several of these methods are named the same as those in `auto_rate` for consistency and have equivalent outcomes, so this allows results to be reordered to the equivalent of that method's results without re-running the `auto_rate` analysis.

The `"row"` and `"rolling"` methods reorder sequentially by the starting row of each regression (`$row` column).

The `"time"` method reorders sequentially by the starting time of each regression (`$time` column).

`"linear"` and `"density"` are essentially identical, reordering by the `$density` column. This metric is only produced by the `auto_rate` `linear` method, so will not work with any other results.

`"rep"` or `"rank"` both reorder by the `$rep` then `$rank` columns. What these represents is context dependent - see **Replicate and Rank columns** section above. Each summary row `rep` and `rank` value is retained unchanged regardless of how the results are subsequently selected or reordered, so this will restore the original ordering after other methods have been applied.

`"rsq"` reorders by `$rsq` from highest value to lowest.

`"intercept"` and `"slope"` reorder by the `$intercept_b0` and `$slope_b1` columns from lowest value to highest.

`"highest"` and `"lowest"` reorder by absolute values of the `$rate.output` column, that is highest or lowest in magnitude regardless of the sign. They can only be used when rates all have the same sign.

`"maximum"` and `"minimum"` reorder by numerical values of the `$rate.output` column, that is maximum or minimum in numerical value taking account of the sign, and can be used when rates are a mix of negative and positive.

Numeric input conversions:

For `convert_rate` objects which contain rates which have been converted from numeric values, the summary table will contain a limited amount of information, so many of the selection or reordering methods will not work. In this case a warning is given and the original input is returned.

Plot:

There is no plotting functionality in `select_rate`. However since the output is a `convert_rate` object it can be plotted. See the **Plot** section in `help("convert_rate")`. To plot straight after a selection operation, pipe or enter the output in `plot()`. See Examples.

More:

This help file can be found online here, where it is much easier to read.

For additional help, documentation, vignettes, and more visit the `respR` website at `https://januarharianto.github.io/respR/`

Value

The output of `select_rate` is a `list` object which retains the `convert_rate` class, with an additional `convert_rate_select` class applied.

It contains two additional elements: `$original` contains the original, unaltered `convert_rate` object, which will be retained unaltered through multiple selection operations, that is even after processing through the function multiple times. `$select_calls` contains the calls for every selection operation that has been applied to the `$original` object, from the first to the most recent. These additional elements ensure the output contains the complete, reproducible history of the `convert_rate` object having been processed.

Examples

```
## Object to filter
 ar_obj <- inspect(intermittent.rd, plot = FALSE) %>%
   auto_rate(plot = FALSE) %>%
   convert_rate(oxy.unit = "mg/L",
                time.unit = "s",
                output.unit = "mg/h",
                volume = 2.379) %>%
   summary()

 ## Select only negative rates
 ar_subs_neg <- select_rate(ar_obj, method = "negative") %>%
   summary()

 ## Select only rates over 1000 seconds duration
 ar_subs_dur <- select_rate(ar_obj, method = "duration", n = c(1000, Inf)) %>%
   summary()

 ## Reorder rates sequentially (i.e. by starting row)
 ar_subs_dur <- select_rate(ar_obj, method = "row") %>%
   summary()

 ## Select rates with r-squared higher than 0.99,
 ## then select the lowest 10th percentile of the remaining rates,
 ## then take the mean of those
 inspect(squid.rd, plot = FALSE) %>%
   auto_rate(method = "linear",
             plot = FALSE) %>%
   convert_rate(oxy.unit = "mg/L",
                time.unit = "s",
                output.unit = "mg/h",
                volume = 2.379) %>%
   summary() %>%
   select_rate(method = "rsq", n = c(0.99, 1)) %>%
   select_rate(method = "lowest_percentile", n = 0.1) %>%
   mean()
```

`squid.rd` *Respirometry data of the squid,* Doryteuthis opalescens

Description

A single experiment on the squid species *Doryteuthis opalescens* in a Loligo Systems swim tunnel and Witrox oxygen probe system. Oxygen was recorded to very low concentrations, making this dataset suitable for determining PCrit. Experiment conducted at Hopkins Marine Station, Stanford University, Pacific Grove, California. Mean temperature, salinity and atmospheric pressure are supplied below to allow for conversion to oxygen concentration units.

Usage

```
squid.rd
```

Format

A data frame object consisting of 2 columns (`$Time` and `$Oxygen`) and 34120 rows (approx 9.5h of data).

Details

- Dissolved oxygen units: `mg/L`
- Time units: `seconds`
- Chamber volume (L): `12.3`
- Specimen wet mass (kg): `0.02141`
- Temperature (°C): `14`
- Salinity: `35`
- Atm. Pressure (bar): `1.013253`

Data kindly supplied by Ben Burford, Hopkins Marine Station, Stanford University.

Author(s)

Ben Burford

subsample	*Subsample a data frame object*

Description

A simple function that subsamples a data frame or numeric vector in order to "thin" large datasets.

Usage

```
subsample(x, n = NULL, length.out = NULL, random_start = FALSE, plot = TRUE)
```

Arguments

`x`	data frame or vector. The data to subsample.
`n`	numeric. Subsample every `n` elements or rows.
`length.out`	numeric. Subsample to a specific length or number of rows.
`random_start`	logical. Defaults to FALSE. If TRUE, randomises the start position from which to start the subsample (applies to `n` input only).
`plot`	logical. Defaults to TRUE. Plots the data. If there are multiple columns in the data frame, only the first two are plotted. Vectors are plotted against a position index.

Details

Two subsampling methods are provided. The `n` input selects every n'th element or row, or alternatively the `length.out` input uniformly subsamples the data to the desired length.

More:
For additional help, documentation, vignettes, and more visit the `respR` website at `https://januarharianto.github.io/respR/`

Value

Returns a subsampled data frame or vector object depending on input.

Examples

```
# Subsample by every 200th row:
subsample(squid.rd, n = 200)

# Subsample to 100 rows:
subsample(sardine.rd, length.out = 100)

# Subsample with random starting position:
subsample(sardine.rd, n = 20, random_start = TRUE)

# Subsample a vector
subsample(sardine.rd[[2]], n = 20)
```

subset_data *Subset a* data.frame, inspect, *or* inspect.ft *object*

Description

subset_data subsets a data.frame, inspect, or inspect.ft object based on a given set of criteria. The function is ideal for passing only selected regions of data to other functions such as calc_rate() and auto_rate(), either by saving the output as a new object or via the use of pipes (%>% or %>%). It is also very useful in analysis of intermittent-flow data, where in a loop each replicate can be extracted and passed to an analytical function such as calc_rate or auto_rate. See examples and vignettes.

Usage

```
subset_data(x, from = NULL, to = NULL, by = "time", quiet = TRUE)
```

Arguments

x	data.frame, inspect, or inspect.ft object. The data from which to produce a subset.
from	numeric. The lower bounds of the subset based on the by input.
to	numeric. The upper bounds of the subset based on the by input.
by	string. "time", "row", or "oxygen". Method by which to apply the from and to inputs.
quiet	logical. Controls if a summary of the output is printed to the console. Default is TRUE.

Details

The function can subset data based on ranges of "time", "oxygen", or "row". For data frames, to subset by "time" or "oxygen" the time data is assumed to be in the first column, and oxygen data in the second column. For inspect() and inspect.ft() objects, the data will have been coerced to this structure already. In these cases the $dataframe element in the output is replaced by the subset, and in inspect.ft the $data element is also subset and replaced. Note for inspect.ft objects, the oxygen data in column 2 will be either out.oxy data or delta.oxy data depending on what was inspected. The function can subset *any* data frame by row.

When multiple columns are present, for example time in column 1, and multiple columns of oxygen data, the subset object will include *all* columns. In the case of subsetting by = "oxygen", subsetting is based on the *first* column of oxygen data only (i.e. column 2), and all subsequent columns are subset between the same rows regardless of oxygen values.

For all methods, if exact matching values of from and to are not present in the data, the closest values are used. For "time" and "row" subsetting, from and to should be in the correct order. No warning or messages are given if the input values are outside those in the data frame. For instance, if to = 100 and there are only 50 rows in the data, the last row (50) will be used instead. The same for from and to time values outside those in the data frame.

For "oxygen" subsetting, from and to are generally interchangeable, and the function will subset data *between* the first and last occurrences (or closest occurrences) of these values. It works best with generally increasing or decreasing oxygen data, and results may vary with other data such as intermittent flow data or those in inspect.ft objects.

Note for inspect and inspect.ft object inputs: after subsetting the locations of any data issues highlighted when the object was originally inspected will no longer be accurate. If these are important, best practice is to subset the original dataframe, and then process the subset through inspect or inspect.ft.

A summary of the subset can be printed to the console if the default quiet = FALSE is changed to TRUE.

More:

For additional help, documentation, vignettes, and more visit the respR website at https://januarharianto.github.io/respR/

Value

Output: If the input is an inspect, or inspect.ft object, the output is an object of the same class containing the subset data. For data.frame inputs the output is a data.table of the subset.

Examples

```
# Subset by time:
x <- subset_data(squid.rd, from = 2000, to = 4000, by = "time")

# Subset by oxygen:
subset_data(sardine.rd, from = 94, to = 91, by = "oxygen")

# Subset by row:
subset_data(flowthrough.rd, from = 10, to = 750, by = "row")

# Subset multiple columns:
# In this case subsetting is based on the first two columns
subset_data(flowthrough.rd, from = 50, to = 600, by = "time")

# Pass (via piping) only a subset of a dataset to inspect() and auto_rate()
subset_data(sardine.rd, from = 94, to = 91, by = "oxygen") %>%
   inspect(time = 1, oxygen = 2) %>%
   auto_rate()
```

---

unit_args *Print examples of unit inputs*

Description

This is a basic function with no inputs. It prints to the console the units that can be used in the functions convert_DO(), convert_MR(), convert_rate(), and convert_rate.ft().

Usage

```
unit_args()
```

Details

Note that some oxygen unit conversions require temperature (`t`), salinity (`S`), and atmospheric pressure (`P`) to be specified.

Note the difference between percent air saturation (`%Air`), where air saturated water is ~100%, and percent oxygen saturation (`%Oxy`), where air saturated water is ~20.946% *oxygen* saturated. In other words, `%Oxy = %Air x 0.20946`.

For most units a fuzzy string matching algorithm is used to accept different formatting styles. For example, `"mg/l"`, `"mg/L"`, `"mgL-1"`, `"mg l-1"`, `"mg.l-1"` are all parsed the same.

`convert_DO()`:

Oxygen concentration or pressure units for `from` *and* `to`*::*
Oxygen concentration units. Should use SI units (`L` or `kg`) for the denominator.
Do *NOT* require `t`, `S` and `P` for conversions:

- `"mg/L"`, `"ug/L"`, `"mol/L"`, `"mmol/L"`, `"umol/L"`, `"nmol/L"`, `"pmol/L"`

Require `t`, `S` and `P` for conversions:

- `"uL/L"`, `"mL/L"`, `"mm3/L"`, `"cm3/L"`, `"mg/kg"`, `"ug/kg"`, `"mol/kg"`, `"mmol/kg"`, `"umol/kg"`, `"nmol/kg"`, `"pmol/kg"`, `"uL/kg"`, `"mL/kg"`, `"ppm"` (i.e. parts per million, equivalent to `mg/kg`).

Percentage saturations (require `t`, `S` and `P`):
`"%Air"` (i.e. % Air Saturation), `"%Oxy"` (i.e. % Oxygen Saturation)
Pressure units (require `t`, `S` and `P`):
`"Torr"`, `"hPa"`, `"kPa"`, `"mmHg"`, `"inHg"`

`convert_rate()` and `convert_rate.ft()`:

Oxygen concentration or pressure units for `oxy.unit`*::*
See above.

Time units for `time.unit` *or as part of* `flowrate.unit`*::*

- `"sec"`, `"min"`, `"hour"`, `"day"`

Volume units for use as part of `flowrate.unit` *(*`convert_rate.ft` *only)::*
For example, in `'ml/min'`, `'L/s'`, etc.

- `"uL"`, `"mL"`, `"L"`

Metabolic rate units:

Combining units for `output.unit` in `convert_rate()` and `convert_rate.ft()`, or for use in `convert_MR()`, must follow these orders:

- Absolute rates: `Oxygen/Time` e.g. `"mg/s"`, `"umol/min"`, `"mL/h"`
- Mass-specific rates: `Oxygen/Time/Mass` e.g. `"mg/s/ug"`, `"umol/min/g"`, `"mL/h/kg"`
- Area-specific rates: `Oxygen/Time/Area` e.g. `"mg/s/mm2"`, `"umol/min/cm2"`, `"mL/h/m2"`

Oxygen amount units:

- `"ug"`, `"mg"`, `"pmol"`, `"nmol"`, `"umol"`, `"mmol"`, `"mol"`, `"uL"`, `"mL"`, `"mm3"`, `"cm3"`

Note `"mm3"` and `"cm3"` (i.e. `cc`) are used in some older publications. These are equivalent to `"uL"` and `"mL"` respectively.

Time units:

- `"sec"`, `"min"`, `"hour"`, `"day"`

Mass units for mass-specific rates:

- `"ug"`, `"mg"`, `"g"`, `"kg"`

Area units for area-specific rates:

- `"mm2"`, `"cm2"`, `"m2"`, `"km2"`

Value

A print out to the console of accepted units

Examples

```
# Run the function:
unit_args()
```

`urchins.rd` *Multi-column respirometry data of the sea urchin,* Heliocidaris Erythrogramma*, including background respiration*

Description

Oxygen consumption data of 16 individual *Heliocidaris erythrogramma* specimens. In addition, there are two columns of background respiration.

Usage

```
urchins.rd
```

Format

A data frame object consisting of one column of time, 16 columns of urchin oxygen consumption (`a` to `p`) and 2 columns of background oxygen consumption (`b1` & `b2`). There are 271 rows of data spanning 45 minutes.

Details

- Dissolved oxygen units: `mg/L`
- Time units: `minutes`
- Volume (L): `1.09`
- Temperature (°C): `20`
- Salinity: `30`
- Atm. Pressure (bar): `1.01`

Author(s)

Januar Harianto

`zeb_intermittent.rd` *Respirometry data of a zebrafish,* Danio rerio

Description

Multiple measurements (106 replicates, plus initial and end background measurements) of oxygen consumption in a zebrafish, *Danio rerio*, obtained using intermittent flow respirometry. Data kindly provided by Davide Thambithurai (University of Glasgow). Note, the data has been injected with random noise, and volume and mass below are not the actual values from the experiment.

Usage

```
zeb_intermittent.rd
```

Format

A data frame object consisting of 2 columns (time and dissolved oxygen) and 79251 rows (approx 22h of data).

Details

- Dissolved oxygen units: `mg/L`
- Time units: `seconds`
- Chamber volume (L): `0.12`
- Specimen wet mass (kg): ‘0.0009
- Temperature (°C): `25`
- Salinity: `0`
- Atm. Pressure (bar): `1.013253`

Replicate structure (Rows - Experiment section):

- `1:4999` - Start background recording
- `5000:5839` - First replicate for MMR (14 mins duration)
- `5840:75139` - 105 further replicates of 11 minutes duration each (660 rows)
- `75140:79251` - End background recording

Each replicate comprises a measurement period (12 minutes for replicate 1, 9 minutes for all others) plus 2 minutes flush.

Author(s)

Davide Thambithurai, University of Glasgow

Index