Package 'forceR'

March 1, 2023

Title Force Measurement Analyses

Description For cleaning and analysis of graphs, such as animal closing force measurements.
'forceR' was initially written and optimized to deal with insect bite force measurements, but can be used for any time series. Includes a full workflow to load, plot and crop data, correct amplifier and baseline drifts, identify individual peak shapes (bites), rescale (normalize) peak curves, and find best polynomial fits to describe and analyze force curve shapes.

Version 1.0.20

Date 2023-02-28

License MIT + file LICENSE

URL https://github.com/Peter-T-Ruehr/forceR

BugReports https://github.com/Peter-T-Ruehr/forceR/issues

Encoding UTF-8

RoxygenNote 7.2.3

Imports dplyr, graphics, grDevices, filesstrings, magrittr, purrr, readr, roll, stats, stringr

Depends R (>= 4.2)

VignetteBuilder knitr

Suggests ggplot2, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Maintainer Peter T. Rühr <peter.ruehr@gmail.com>

LazyData true

NeedsCompilation no

Author Peter T. Rühr [aut, cre] (<https://orcid.org/0000-0003-2776-6172>), Alexander Blanke [ctb] (<https://orcid.org/0000-0003-4385-6039>)

Repository CRAN

Date/Publication 2023-03-01 08:22:36 UTC

R topics documented:

`amp_drift_corr` *Charge Amplifier Drift Correction*

Description

Removes the systemic, asymptotical drift of charge amplifiers with resistor-capacitor (RC) circuits.

Usage

```
amp_drift_corr(
  filename,
  tau = 9400,
  res.reduction = 10,
  plot.to.screen = FALSE,
  write.data = FALSE,
  write.PDFs = FALSE,
  write.logs = FALSE,
  output.folder = NULL,
  show.progress = FALSE
)
```

Arguments

`filename`	Path to file on which amplifier drift correction should be performed.
`tau`	Numeric time constant of charge amplifier in the same time unit as the measurement data. Default: `9400`
`res.reduction`	A numeric value to reduce the number of time steps by during plotting. Speeds up the plotting process and reduces PDF size. Has no effect on the results, only on the plots. Default: `10`.
`plot.to.screen`	A logical value indicating if results should be plotted in the current R plot device. Default: `FALSE`.
`write.data`	A logical value indicating if drift-corrected file should be saved. If yes, it will be saved in `output.folder`. Default: `FALSE`.
`write.PDFs`	A logical value indicating whether results should be saved as PDFs. Does not slow down the process as much as printing to the R plot device and is considered necessary to quality check the results. If yes, it will be saved in `output.folder/PDFs`. Default: `FALSE`.
`write.logs`	A logical value indicating whether a log file with information on the method and values used to correct the amplifier drift should be saved. Is considered necessary for reproducibility. If yes, it will be saved in `output.folder/logs`. Default: `FALSE`.
`output.folder`	Path to folder where data, PDF and log files should be stored.
`show.progress`	A logical value indicating if progress should be printed to the console. Slows down the process. Default: `FALSE`.

Details

forceR generally expects file names to start with a leading number specifying the measurement number (E.g. "0001_G_maculatus.csv"). The number ("0001") is used to keep data files, log files, and PDF files of the same measurement associated with each other.

The input file should be in the following format:

`t`	`y`
`t.1`	`y.2`

```
...     ...
t.n     y.n
```

Value

Returns a tibble containing the amplifier drift-corrected data in the following format

```
 t       y
t.1     y.2
...     ...
t.n     y.n
```

Examples

```
# define file for amplifier drift correction
filename <- forceR_example(type = "raw")

# Run amplifier drift correction without saving files or printing to screen:
file.ampdriftcorr <- amp_drift_corr(filename = filename,
                                    tau = 9400,
                                    res.reduction = 10,
                                    plot.to.screen = FALSE,
                                    write.data = FALSE,
                                    write.PDFs = FALSE,
                                    write.logs = FALSE,
                                    output.folder,
                                    show.progress = FALSE)
# file.ampdriftcorr

# Run amplifier drift correction with saving files and printing to screen:
#    - commented out to pass package tests
# file.ampdriftcorr <- amp_drift_corr(filename = filename,
#                                     tau = 9400,
#                                     res.reduction = 10,
#                                     plot.to.screen = TRUE,
#                                     write.data = TRUE,
#                                     write.PDFs = TRUE,
#                                     write.logs = TRUE,
#                                     output.folder = "./ampdriftcorr",
#                                     show.progress = TRUE)
#
# file.ampdriftcorr
```

---

avg_peaks *Average Curves per Group*

Description

Calculates mean curve shape per group (here: species) and rescales result on the y axis to range from 0 to 1.

Usage

```
avg_peaks(df, path.data = NULL)
```

Arguments

df	The resulting tibble of the function red_peaks_100(). See ?red_peaks_100 for more details.
path.data	A string character defining where to save the results. If NULL, data will not be saved to disk. Default: NULL.

Value

This function returns a tibble made of three columns: species containing the species names, index ranging from 1 to 100 for each species, and force.norm.100 containing the averaged and rescaled curve of each species.

Examples

```
# Using the forceR::df.all.200.tax dataset:

# calculate mean curves per species
peaks.df.100.avg <- avg_peaks(df = forceR::peaks.df.norm.100,
                              path.data = NULL)

# plot averaged normalized curves per species
require(ggplot2)
ggplot(peaks.df.100.avg,
          aes(x = index ,
              y = force.norm.100.avg,
              colour=species)) +
  geom_line()
```

---

baseline_corr *Automatic or Manual Baseline Correction of Time Series*

Description

If baseline (zero-line) of measurement is unstable (e.g. due to temperature fluctuations, wind, ...), the baseline needs to be continually adjusted throughout the measurement. This script allows an automatic adjustment of the baseline. The automatic approach invokes a sliding window, during which the 'minimum' within each sliding window is stored. A 'minimum' is defined by the quantile.size: if set to 0.05, the value below which only 5% of the measurement data within the

sliding window lies, is treated as the current window's minimum. This prevents the treatment of potential artifacts as minima. In a second iteration, another sliding window calculates the average of these 'minima'. The resulting values are subtracted from the original time series. This approach works well for time series with relatively short peaks. If the automatic approach does not yield acceptable results, an interactive manual approach to correct the baseline can be performed instead.

Usage

```
baseline_corr(
  filename,
  corr.type = "auto",
  window.size.mins = 1000,
  window.size.means = NULL,
  quantile.size = 0.05,
  y.scale = 0.5,
  res.reduction = 10,
  Hz = 100,
  plot.to.screen = FALSE,
  write.data = FALSE,
  write.PDFs = FALSE,
  write.logs = FALSE,
  output.folder = NULL,
  show.progress = FALSE
)
```

Arguments

`filename`	A character string containing the full path to the measurement file that needs correction. See Details for info on what the file should look like.
`corr.type`	Character string defining the desired mode of baseline correction. One of `"auto"` or `"manual"`. Default: `"auto"`
`window.size.mins`	A numeric value for the size of the search window to find minima in. Should be in the same time unit as the measurement. Longer peaks require higher values, shorter peaks require smaller values. Default: `1000`.
`window.size.means`	A numeric value for the size of the window to average the minima in. Should be in the same time unit as the measurement. By default (`NULL`), the same value as specified for `window.size.mins` is used.
`quantile.size`	A numerical value between `0` and `1` to define the quantile which is treated as the 'minimum' of a sliding window. Default: `0.05`.
`y.scale`	A numeric value to reduce the y-axis range during plotting. This simplifies the manual placement of the points during the manual correction procedure.
`res.reduction`	A numeric value to reduce the number of time steps by during plotting. Speeds up the plotting process and reduces PDF size. Has no effect on the results, only on the plots. Default: `10`.

Hz	A numeric value to reduce sampling frequency for temporary analyses. This works as a smoothing filter during temporary analyses and does not reduce the actual sampling frequency of the data. Default: 100.
plot.to.screen	A logical value indicating if results should be plotted in the current R plot device. Default: FALSE.
write.data	A logical value indicating if drift-corrected file should be saved. If yes, it will be saved in output.folder. Default: FALSE.
write.PDFs	A logical value indicating whether results should be saved as PDFs. Does not slow down the process as much as printing to the R plot device and is considered necessary to quality check the results. If yes, it will be saved in output.folder/PDFs. Default: FALSE.
write.logs	A logical value indicating whether a log file with information on the method and values used to correct the baseline drift should be saved. Is considered necessary for reproducibility. If yes, it will be saved in output.folder/logs. Default: FALSE.
output.folder	Path to folder where data, PDF and log files should be stored. Default: NULL.
show.progress	A logical value indicating if progress should be printed to the console. Default: FALSE.

Details

forceR generally expects file names to start with a leading number specifying the measurement number (E.g. "0001_G_maculatus.csv"). The number ("0001") is used to keep data files, log files, and PDF files of the same measurement associated with each other.

The input files should to be in the following format:

t	y
t.1	y.2
...	...
t.n	y.n

In case there are more than two columns, only the first two columns will be used. If the first two columns are not named 't' and 'y', they will be renamed.

Value

Returns a tibble containing the amplifier baseline drift-corrected data in the following format

t	y
t.1	y.2
...	...
t.n	y.n

Examples

```
#'############ AUTOMATIC MODE
# define file to apply the baseline drift correction to
filename <- forceR_example(type = "ampdriftcorr")

# run automatic baseline drift corrections without saving files or
#   printing to screen:
file.baseline_corr <- baseline_corr(filename = filename,
                                    corr.type = "auto",
                                    window.size.mins = 1000,
                                    window.size.means = NULL,
                                    quantile.size = 0.05,
                                    y.scale = 0.5,
                                    res.reduction = 10,
                                    Hz = 100,
                                    plot.to.screen = FALSE,
                                    write.data = FALSE,
                                    write.PDFs = FALSE,
                                    write.logs = FALSE,
                                    show.progress = FALSE)

file.baseline_corr

#'############ MANUAL MODE

# define file to apply the baseline drift correction to
filename <- forceR_example(type = "ampdriftcorr")

# run interactive baseline drift corrections with saving files and
#   printing to screen:  - out-commented to pass package tests

# file.baseline_corr <- baseline_corr(filename = filename,
#                                     corr.type = "manual",
#                                     window.size.mins = 1000,
#                                     window.size.means = NULL,
#                                     quantile.size = 0.05,
#                                     y.scale = 0.5,
#                                     res.reduction = 10,
#                                     Hz = 100,
#                                     plot.to.screen = TRUE,
#                                     output.folder = "./baselinecorr",
#                                     show.progress = TRUE)

# file.baseline_corr
```

`classifier` *Classifier*

Description

Start and end time values of the 5 strongest peaks per species of `df.all` with the names of the measurements in which they occur.

Usage

```
classifier
```

Format

A data frame with 24 rows and 5 columns:

species species names

specimen specimen names

measurement measurement names

amp amplifier values, in V/N

lever.ratio ration of on-lever to out-lever of mesaurement setup

Details

Result of `rescale_peaks()`.

---

convert_measurement *Converts LJStream *.dat file to standard time series.*

Description

Converts LJStream *.dat file to standard time series.

Usage

```
convert_measurement(file, path.data = NULL, collect_garbage = FALSE)
```

Arguments

`file`	File path to raw measurement (*.dat file).
`path.data`	A string character defining where to save the results. If `NULL`, data is not stored in a file. Default: `NULL`.
`collect_garbage`	Logical. If `TRUE`, then the `gc()` command will be run silently to try to clean up memory. This may help when running `convert_measurement` in a loop, even though memory cluttering cannot be fully prevented. If such a loop crashes, the loop should be split into several separate loops to convert all files. Default: `FALSE`.

Value

Returns and, if `path.data is not NULL`, saves data in csv-format in `path.data`.

The output tibble has the following format:

t	y
t.1	y.1
...	...
t.n	y.n

Examples

```
# get file path of forceR example file
filename <- forceR_example(type = "LJStream")
file.converted <- convert_measurement (file = filename,
                       path.data = NULL)
file.converted
```

`correct_peak` *Manually Correct Single Peak*

Description

Interactive correction of a single peak.

Usage

```
correct_peak(
  df.peaks,
  df.data,
  measurement,
  peak,
  additional.msecs = 500,
  path.data = NULL
)
```

Arguments

`df.peaks`	The resulting tibble of the function `find_peaks()`. See `?find_peaks` for more details.
`df.data`	A data frame or tibble in the below format. The columns `t` (time), `force`, `measurement`, and `specimen`. (measurement ID) must be present. This will usually be the same table that was used before in `find_peaks()`.

`measurement` A character string defining the measurement ID (e.g. measurement number) of the peak to be corrected. Must be congruent with the respective measurement ID within `df.peaks` and `df`.

`peak` A numerical value defining the peak to be corrected.

`additional.msecs` A numerical value defining the time before and after the originally defined window of the peak to be corrected. Higher values allow defining start and end points further away from the original start and end points. Default: `500`. @param path.data A string character defining where to save the result and log file. If `NULL` (default), data is not stored in files. Log files cannot be retrieved in this case.

`path.data` A string character defining where to save the results. If `NULL` (default), data is not stored in a file.

Details

When running the function, you will be prompted to select the actual start and end of the current `peak` of the current `measurement`. If more or less than exactly 2 points are defined, the process is terminated.

Value

Changes values within `df.peaks` and returns the changed tibble.

`df.data` **needs to contain the following columns:**

t	force	measurement
t.1	force.1	measurement.1
...	...	...
t.n	force.n	measurement.m

Examples

```
# Using the forceR::df.all.200.tax dataset:

# This function needs user input.
peaks.df <- correct_peak(df.peaks = forceR::peaks.df,
                         df.data = forceR::df.all.200.tax,
                         measurement = "m_01",
                         peak = 1,
                         additional.msecs = 5)
```

crop_measurement *Crop Time Series*

Description

Interactive function to crop a time series.

Usage

```
crop_measurement(file, path.data = NULL)
```

Arguments

file	File path to measurement.
path.data	A string character defining where to save the results. If NULL, data is not stored in a file. Default: NULL.

Details

Select points at start and end of desired part of measurements. Only the last two points will be taken into account to allow the user to correct erroneous clicks.

If a measurement contains two distinct regions of bites with a lot of unnecessary data and/or measurement artefacts in-between (such as user-made peaks), I recommend to manually copy the RAW data files, give the copy a new measurement number (as if it was actually a separate measurement), and crop the distinct parts containing actual bites separately from the two copies of the file. For more distinct regions, create more copies.

I recommend to not crop the files too much in case baseline corrections are needed later, because then the baseline_corr() function will not be able to figure out where the actual baseline might be. Leaving several seconds before and after the first and last bite of a series will prevent such problems.

Value

Returns and, if path.data is not NULL, saves data in csv-format in path.data.

The tibble has the following format:

t	y
t.1	y.1
...	...
t.n	y.n

Examples

```
# get file path of forceR example file
filename <- forceR_example(type = "raw")
```

```
# # crop file without storing result as file - out-commented to pass package tests
# file.cropped <- crop_measurement(file = filename,
#                    path.data = NULL)

# file.cropped
```

`df.all` *Simulated Force Measurements with Taxonomic Info.*

Description

A series of 24 measurements with six simulated peaks each. Simulated using the `simulate_bites()` function . Fits the `classifier` sheet that is also part of the `forceR` package.

Usage

```
df.all
```

Format

A data frame with 9,600 rows and 3 columns:

t time, in ms

y measured values, in V

measurement measurement names

`df.all.200` *Simulated Time Series - e.g. Bite Force Measurements*

Description

A series of 24 measurements with six simulated peaks each. Simulated using the `simulate_bites()` function. Reduced to a sampling frequency of 200 Hz with `reduce_frq()`. Fits the `classifier` sheet that is also part of the `forceR` package.

Usage

```
df.all.200
```

Format

A data frame with 1,944 rows and 3 columns:

t time, in ms

y measured values, in V

measurement measurement names

df.all.200.tax *Simulated Force Measurements with Taxonomic Info.*

Description

A series of 24 measurements with six simulated peaks each. Simulated using the `simulate_bites()` function and translated into force values with `y_to_force()`. Fits the `classifier` sheet that is also part of the `forceR` package.

Usage

```
df.all.200.tax
```

Format

A data frame with 1,944 rows and 5 columns:

species species names

specimen specimen names

measurement measurement names

t time, in ms

force measured values, in N

find_best_fits *Find Best Polynomial Fits for Curves*

Description

Calculates best model fits for all curves based on AIC criterion. The function fits polynomial functions with 1 to 20 coefficients and uses the Akaike Information Criterion (AIC) to evaluate the goodness of the fits. A model is considered a good fit, when the percentage of change from one model to the next (e.g. a model with 6 coefficients to a model with 7 coefficients) is, e.g. `< 5%` when `threshold = 5`. The first for models meeting this criterion are plotted as colored graphs and the AICs of these models are visualized in a second plot for each curve. All first four coefficients per curve that fulfill the criterion are stored and in the end, a histogram of how often which coefficients were good fits is plotted as well. The function returns the numerical value of the coefficient that fulfilled the criterion of a good fit in most curves.

Usage

```
find_best_fits(
  df,
  degrees = 1:20,
  threshold = 5,
  zero_threshold = NULL,
  plot.to.screen = FALSE,
  path.data = NULL,
  path.plots = NULL,
  show.progress = FALSE
)
```

Arguments

df	The resulting tibble of the function avg_peaks(). See below for more details.
degrees	Numerical vector of polynomial degrees to test. Cannot be infinitely high - if two high, throws error: 'degree' must be less than number of unique points. Default: 1:20.
threshold	Percentage of AIC change compared to previous degree to fit the good-fit-criteria (s.a.). Default: 5.
zero_threshold	Either numerical or NULL: If numerical, the function checks if the graph of the current model starts and ends near zero, e.g. below 0.2 if zero_threshold = 0.2. Default: NULL.
plot.to.screen	A logical value indicating if results should be plotted in the current R plot device. Default: FALSE.
path.data	A string character defining where to save the results. If NULL, data is not stored in a file. Default: NULL.
path.plots	A string character defining where to save the plots. If NULL, plots will not be saved to PDF files. Default: NULL.
show.progress	A logical value indicating if progress should be printed to the console. Default: FALSE.

Details

#' This function expects a tibble made of three columns as df: species containing the species names, index numerical column, e.g. time (but can be arbitrary continuous unit), for each species, and force.norm.100 containing the averaged and rescaled curve of each species.

Value

Returns the a numerical value representing the number of coefficient that was most often under the first 4 models that were followed by an AIC-change <= 5% by the next model. Additionally, plots showing the model fits and a histogram of the coefficients that met the 5%-criterion can be plotted to the plot device or saved as PDFs in path.plots.

Examples

```
# Using the forceR::peaks.df.100.avg dataset:

# find smallest polynomial degree that best describes all curves
best.fit.poly <- find_best_fits(df = forceR::peaks.df.100.avg)

best.fit.poly
```

`find_strongest_peaks` *Find Peaks*

Description

Identifies peaks in a first iteration and optimizes the starts and ends of the strongest peaks per species in a second iteration.

Usage

```
find_strongest_peaks(
  df,
  no.of.peaks = 5,
  initial.threshold = 0.05,
  slope.length.start = 5,
  slope.length.end = 5,
  slope.thresh.start = 0.02,
  slope.thresh.end = 0.02,
  path.data = NULL,
  path.plots = NULL,
  show.progress = FALSE
)
```

Arguments

`df`	A data frame or tibble in the below format. The columns `t` (= time), `force` and `measurement` (= measurement ID) must be present.
`no.of.peaks`	A numeric value defining how many peaks per `species` (not per `measurement`") should be identified. The function will always return the strongest peaks. Default: `5`
`initial.threshold`	A numeric value defining the threshold (in % of the maximum force of the measurement) that is used during the first iteration. Default: `0.05`
`slope.length.start`	A numeric value defining the window size (in time steps) of slope calculation for peak starts during the second iteration. Default: `5`

`slope.length.end`
: A numeric value defining the window size (in time steps) of slope calculation for peak ends during the second iteration. Default: `5`

`slope.thresh.start`
: A numeric value defining the threshold at which to stop the sliding window and save the current time point as the actual start of the current peak. Default: `0.04`

`slope.thresh.end`
: A numeric value defining the threshold at which to stop the sliding window and save the current time point as the actual end of the current peak. Default: `0.04`

`path.data`
: A string character defining where to save the results. If `NULL` (default), data is not stored in a file. Default: `NULL`.

`path.plots`
: A string character defining where to save the plots. Default: `NULL`.

`show.progress`
: A logical value indicating if progress should be printed to the console. Default: `FALSE`.

Details

The input data frame `df` needs to contain the following columns:

t	force	measurement
t.1	force.1	measurement.1
...	...	...
t.n	force.n	measurement.m

Value

Creates a tibble in the following format and saves it as a CSV-file: The column `species` contains one species per row
(`species.1 ... species.n`).

The column `measurements` contains as many measurements as `no.of.peaks`, separated by ';':
(`measurement.1; ...; measurements.no.of.peaks`).

The column `starts` contains as many peak starts as `no.of.peaks`, separated by ';':
(`start.1; ...; start.no.of.peaks`).

The column `ends` contains as many peak ends as `no.of.peaks`, separated by ';':
(`end.1; ...; end.no.of.peaks`).

Examples

```
require(dplyr)
# Using the forceR::df.all.200.tax dataset:

# reduce dataset (only rows 40 to 95 of species A (containing data of
  # measurement 1 and 2 (m_01 and m_02)))
df.all.200.tax_filtered <- forceR::df.all.200.tax[40:95, ] %>%
  filter(species == "species_A")

# find the 4 strongest peaks
```

```
peaks.df <- find_strongest_peaks(df = df.all.200.tax_filtered,
                                 no.of.peaks = 4)

# plot results (three peaks in measurement 1, 1 peak in measurement 2):
# plot_peaks(df.peaks = peaks.df,
#            df.data = df.all.200.tax_filtered,
#            additional.msecs = 20)
```

`forceR_example` *Get path to forceR example*

Description

forceR comes with example files of short bite force measurements. The files are stored in `forceR's inst/extdata` folder, and this function returns the path to that folder or one of the files so they can be used in examples.

Usage

```
forceR_example(type = "folder")
```

Arguments

`type`	A character string (either `"folder"`, `"raw"`, or `"ampdriftcorr"`) defining if the path returned be the function should point to one of the the example files or the folder containing them. Default: `"folder"`.

Value

If `type = "folder"`: returns the file path to the folder containing `BF_raw.csv` and `BF_ampdriftcorr.csv`.

If `type = "LJStream"`: returns the file path to `BF_raw.csv`, which contains a short bite force raw measurement.

If `type = "raw"`: returns the file path to `BF_raw.csv`, which contains a short bite force raw measurement.

If `type = "ampdriftcorr"`: returns the file path to `BF_ampdriftcorr.csv`, which contains a short bite force raw measurement where the amplifier drift as been corrected for by the amp_drift_corr() funcion.

load_mult *Load Multiple Measurements*

Description

Loads multiple measurements.

Usage

```
load_mult(folder, columns = c(1:2), show.progress = FALSE)
```

Arguments

folder	Character string containing the path to the measurements.
columns	A vector of column numbers. The first entry will be used as the x-axis values, the second entry as y-axis values. All other columns will be ignored. Default: c(1,2).
show.progress	A logical value indicating if progress should be printed to the console. Default: FALSE.

Details

The input files need to be in the following format (even though column names do not matter):

t	y
t.1	y.1
...	...
t.n	y.n

All columns except the first two are removed.

Value

Returns a tibble in the format

t	y	filename
t.1	y.1	...
...	...	...
t.n	y.n	...

Examples

```
# store name of folder that contains files
input.folder <- forceR_example(type = "folder")
```

```
# load a mutiple files
df.all <- load_mult(folder = input.folder,
                    columns = c(1:2),
                    show.progress = FALSE)
```

`load_single` *Load single measurement*

Description

Loads a single measurement.

Usage

```
load_single(filename, columns = c(1:2))
```

Arguments

`filename`	Character string containing the path to measurement file.
`columns`	A vector of column numbers. The first entry will be used as the x-axis values, the second entry as y-axis values. All other columns will be ignored. Default: `c(1,2)`.

Details

#' The input files need to be in the following format (even though column names do not matter):

t	y
t.1	y.1
...	...
t.n	y.n

All columns except the first two are removed.

Value

A tibble with two columns named "t" and "y".

Examples

```
# Store filename
filename <- forceR_example(type="raw")

df.1 <- load_single(filename,
                    columns = c(1:2))
```

models *Polynomial Models Describing Average Peak Shapes.*

Description

Extracted from `df.all`.

Usage

```
models
```

Format

A list with 4 named entries, each containing the 4-degree polynomial model that describes the average peak shape of the given species.

peaks.df *Starts and Ends of the 5 Strongest Peaks*

Description

Start and end time values of the 5 strongest peaks per species of `df.all` with the names of the measurements in which they occur.

Usage

```
peaks.df
```

Format

A data frame with 4 rows and 4 columns:

species species names

measurements measurement names

starts start values, in ms

ends end values, in ms

Details

Result of `rescale_peaks()`.

peaks.df.100.avg *Average Peak Shapes per Species*

Description

Normalized force values describing the average shape of the 5 strongest peaks per species of `df.all`

Usage

```
peaks.df.100.avg
```

Format

A data frame with 400 rows and 3 columns:

species species names

index values from 1:100 in each species

force.norm.100.avg normalized values reduced to 100 observations per species, in N

peaks.df.norm *Normalized Peak Shapes*

Description

Normalized force values describing the shapes of all 5 strongest peaks per species of `df.all`

Usage

```
peaks.df.norm
```

Format

A data frame with 223 rows and 6 columns:

measurement measurement names

peak peak numbers

t.norm time values from 0:1 in each measurement, in ms

force.norm force values from 0:1 in each measurement, in N

species species names

specimen specimen names

Details

Result of `rescale_peaks()`.

peaks.df.norm.100 *Normalized Peak Shapes with 100 Observations*

Description

Normalized force values describing the shapes of all 5 strongest peaks per species of `df.all`, reduced to 100 observations per measurement.

Usage

```
peaks.df.norm.100
```

Format

A data frame with 2000 rows and 6 columns:

species species names

measurement measurement names

specimen specimen names

peak peak numbers

index values from 1:100 in each measurement

force.norm.100 normalized values reduced to 100 observations per measurement, in N

Details

Result of `rescale_peaks()`.

peak_duration_max_force *Peak Duration and Maximum Force*

Description

Calculate duration and maximum force for each individual peak.

Usage

```
peak_duration_max_force(
  df.peaks,
  df.data,
  path.data = NULL,
  show.progress = FALSE
)
```

Arguments

df.peaks	The resulting tibble of the function find_peaks(). See ?find_peaks for more details.
df.data	A data frame or tibble in the below format. The columns t (time), force, measurement, and specimen. (measurement ID) must be present. This will usually be the same table that was used before in find_peaks().
path.data	A string character defining where to save the results. If NULL (default), data is not stored in a file.
show.progress	A logical value indicating if progress should be printed to the console. Default: FALSE.

Value

Changes values within df.peaks and returns the changed tibble.

df.data needs to contain the following columns:

t	force	measurement
t.1	force.1	measurement.1
...	...	...
t.n	force.n	measurement.m

Examples

```
# Using the forceR::df.all.200.tax dataset:

# This function needs user input.
peaks.df <- correct_peak(df.peaks = forceR::peaks.df,
                         df.data = forceR::df.all.200.tax,
                         measurement = "m_01",
                         peak = 1,
                         additional.msecs = 5)
```

---

peak_to_poly *Convert Time Series to Polynomial*

Description

Convert Time Series to Polynomial

Usage

```
peak_to_poly(df, coeff, path.data = NULL, show.progress = FALSE)
```

Arguments

df	The resulting tibble of the function avg_peaks(). See ?avg_peaks for more details.
coeff	A numerical value indicating the number of coefficients the model used to fit on the time series data should have.
path.data	A string character defining where to save the results as *.csv and *.R. If NULL, data is not stored in files. Default: NULL.
show.progress	A logical value indicating if progress should be printed to the console. Default: FALSE.

Value

A list with the length equal to the number of unique species within df containing the fitted models.

Examples

```
# Using the forceR::peaks.df.100.avg dataset:

# define the number of coefficients the polynomial models should have
number_of_coeffs = 4

# convert curves to polynomial models
models <- peak_to_poly(df = forceR::peaks.df.100.avg,
                       coeff = number_of_coeffs)

models
```

---

plot_measurement *Plot raw measurement*

Description

Plots a time series.

Usage

```
plot_measurement(file, columns = c(1:2))
```

Arguments

file	File path to measurement.
columns	A vector of column numbers. The first entry will be used as the x-axis values, the second entry as y-axis values. All other columns will be ignored. Default: c(1,2).

Details

#' The input files need to be in the following format (even though column names do not matter):

t	y
t.1	y.1
...	...
t.n	y.n

Value

Creates a plot in the current plot device.

Examples

```
filename = forceR_example(type = "raw")
plot_measurement(filename)
```

`plot_peaks` *Plot Peaks*

Description

Plots the peaks identified by the function `find_peaks()`.

Usage

```
plot_peaks(
  df.peaks,
  df.data,
  additional.msecs = 2000,
  plot.to.screen = TRUE,
  path.plots = NULL,
  show.progress = FALSE
)
```

Arguments

`df.peaks`	df.peaks The resulting tibble of the function `find_peaks()`. See `?find_peaks` for more details.
`df.data`	A data frame or tibble in the below format. The columns `t` (time), `force` and `measurement` (measurement ID) must be present. This will usually be the same table that was used before in `find_peaks()`.
`additional.msecs`	A numeric value indicating how many m.secs before and after the actual peak curve should be plotted. Default: `2000`
`plot.to.screen`	A logical value indicating if results should be plotted in the current R plot device. Default: `TRUE`.

path.plots	A string character defining where to save the plots. If NULL, plots will not be saved to PDF files. Default: NULL
show.progress	A logical value indicating if progress should be printed to the console. Default: FALSE.

Details

df.peaks at least needs to contain the following columns:

measurements | starts | ends | | :—-: | :—-: | :—-: | :—-: | | measurements.1 | starts.1 | ends.1 | | ... | ... | ... | | measurements.n | starts.m | ends.m |

Check forceR::peaks.df to see an example tibble.

df.data at least needs to contain the following columns:

t	force	measurement
t.1	force.1	measurement.1
...	...	...
t.n	force.n	measurement.m

Check forceR::df.all.200.tax to see an example tibble.

Value

Plots one graph per peak curve and, if plot.to.pdf == TURE, saves all peak curves as one PDF at path.plots.

Examples

```
# Using the first row of forceR::peaks.df and the forceR::df.all.200.tax dataset:

# plot peaks
plot_peaks(df.peaks = forceR::peaks.df[1, ],
           df.data = forceR::df.all.200.tax,
           additional.msecs = 20) # instead of the default (2000) because of
                                  # the highly downsampled example dataset.
```

---

print_progress *Print progress*

Description

Prints loop progress in [%] to console.

Usage

```
print_progress(current, end)
```

Arguments

current	Numeric value of current loop iteration.
end	Numeric value number last loop iteration.

Value

Prints percentage of loop progress to console.

---

reduce_frq *Reduce Sampling Frequency*

Description

Reduces the sampling frequency to a certain Hz value. If the desired frequency is smaller than the original frequency, the data remains unchanged.

Usage

```
reduce_frq(df, Hz = 200, measurement.col = NULL)
```

Arguments

df	Data frame or tibble in the below mentioned format.
Hz	Numeric value of desired frequency. Default 200
measurement.col	Character string. If measurement.col is not defined, the whole input data frames will be treated as if it was just one single time series. This is okay for data frames like that indeed only contain one time series, but for data frames with multiple time series, a grouping column needs to be defined. Default: NULL

Details

The input data frame or tibble should have the following format:

t	y
t.1	y.1
...	...
t.n	y.n

or, if measurement.col is not NULL, then

t	y	measurement.col
t.1	y.1	...
...	...	...
t.n	y.n	...

Since, when not NULL, the measurement.col is called by its character string, the position of the column does not matter, except it must not be among the first two columns which are reserved for t and y.

All columns except the first two are removed. Values in t are expected to be in m.secs.

Value

Returns a tibble reduced to the desired frequency in the following format:

t	y
t.1	y.1
...	...
t.n	y.n

or, if measurement.col is not NULL, then

t	y	measurement.col
t.1	y.1	...
...	...	...
t.n	y.n	...

Examples

```
require(dplyr)
# Using the forceR::df.all dataset that was
# simulated with forceR::simulate_bites()

# reduce sampling frequency to 200 Hz
df.all.200 <- reduce_frq(df = df.all,
                         Hz = 200,
                         measurement.col = "measurement")

plot(df.all.200 %>%
       filter(measurement == "m_02") %>%
       select(t, y),
     type = "l", col = "black")
lines(df.all.200 %>%
       filter(measurement == "m_01") %>%
       select(t, y),
     type = "l", col = "blue")
```

---

red_peaks_100 *Reduce Peaks*

Description

Reduces curves to 100 observations per peak.

Usage

```
red_peaks_100(
  df,
  plot.to.screen = FALSE,
  path.data = NULL,
  path.plots = NULL,
  show.progress = FALSE
)
```

Arguments

df	The resulting tibble of the function rescale_peaks(). The columns species, specimen, measurement, peak, and force.norm must be present. See ?rescale_peaks for more details.
plot.to.screen	A logical value indicating if results should be plotted in the current R plot device. Default: FALSE.
path.data	A string character defining where to save the results. If NULL, data will not be saved to disk. Default: NULL.
path.plots	A string character defining where to save result plots. If NULL, plots will not be saved to disk. Default: NULL.
show.progress	A logical value indicating if progress should be printed to the console. Default: FALSE.

Value

This function returns a tibble with a similar format as df, but the columns t, force, t.norm and force.norm are replaced by the columns index, ranging from 1 to 100, and force.norm.100, containing the rescaled force data ranging from 0 to 1. Since the time series has been reduced to 100 observations, this tibble will always contain 100 rows per peak.

df needs to contain the following columns:

force	measurement	peak
force.norm.norm.1	measurement.1	peak.1
...	...	...
force.norm.n	measurement.m	peak.n

Examples

```
# Using the forceR::df.all.200.tax dataset:
```

```
peaks.df.norm.100 <- red_peaks_100(df = forceR::peaks.df.norm,
                                   path.data = NULL,
                                   path.plots = NULL,
                                   show.progress = FALSE)

peaks.df.norm.100
```

`rescale_peaks` *Rescale Peaks*

Description

Rescales time series in x and y to values ranging from 0 to 1.

Usage

```
rescale_peaks(
  df.peaks,
  df.data,
  plot.to.screen = FALSE,
  path.data = NULL,
  show.progress = FALSE
)
```

Arguments

`df.peaks`	The resulting tibble of the function `find_strongest_peaks()`. See `?find_strongest_peaks` for more details.
`df.data`	A data frame or tibble in the below format. The columns `t` (time), `force` and `measurement` (measurement ID) must be present. This will usually be the same table that was used before in `find_strongest_peaks()`.
`plot.to.screen`	A logical value indicating if results should be plotted in the current R plot device. Default: `FALSE`.
`path.data`	A string character defining where to save the results. If `NULL`, data will not be saved to disk. Default: `NULL`.
`show.progress`	A logical value indicating if progress should be printed to the console. Default: `FALSE`.

Details

`df.peaks` at least needs to contain the following columns:

species	measurements	starts	ends
species.1	measurements.1	starts.1	ends.1
...	...	...	...
species.n	measurements.n	starts.m	ends.m

Check `forceR::peaks.df` to see an example tibble.

`df.data` at least needs to contain the following columns:

t	force	measurement
t.1	force.1	measurement.1
...	...	...
t.n	force.n	measurement.m

Check `forceR::df.all.200.tax` to see an example tibble.

Value

This function returns a tibble in the same format as `df`, but with the additional columns `t.norm` and `force.norm` which will contain the rescaled time and force data both ranging from 0 to 1.

Examples

```
# Using the forceR::df.all.200.tax and forceR::df.all.200.tax datasets:

# rescale bites
peaks.df.norm <- rescale_peaks(df.peaks = forceR::peaks.df,
                               df.data = forceR::df.all.200.tax,
                               plot.to.screen = FALSE,
                               path.data = NULL,
                               show.progress = FALSE)


# maximum values of time and force both range from 0 - 1:
range(peaks.df.norm$t.norm)
range(peaks.df.norm$force.norm)
```

`rescale_to_range` *Scale data series to new minimum and maximum*

Description

Maps a series of numeric values to a new range defined by minimum (from) and maximum (to).

Usage

```
rescale_to_range(data, from, to)
```

Arguments

`data`	numeric vector containing the data to be scaled
`from`	minimum of new range
`to`	maximum of new range

Value

numeric vector with scaled data

Examples

```
rescale_to_range(data = 1:10,
                 from = 1,
                 to = 100)
```

`simulate_bites` *Simulate bites*

Description

Simulates either sinusoidal or plateau-like bites.

Usage

```
simulate_bites(
  no.of.bites = 5,
  length.of.bite = 1000,
  length.of.series = 10000,
  max.y = 1,
  max.y.jit = NULL,
  peak.pos = 50,
  slope.perc.start = 10,
  slope.perc.end = slope.perc.start,
  jit = NULL,
  bite.type = "sin",
  plot = TRUE
)
```

Arguments

`no.of.bites`	Number of bites in time series. Default: `5`.
`length.of.bite`	Length of each bite. Default: `1000`.
`length.of.series`	Length of the whole time series. Default: `10000`.
`max.y`	Maximum y value. Default: `1`.
`max.y.jit`	Jitter above and below maximum y value in `[%]` of maximum y value. Default: `NULL`.
`peak.pos`	Position (in percent) of peak within peak curve. Only applies to `bite.type = "sin` (sinusoidal bites.) Default: same as `50`.
`slope.perc.start`	Percentage of how much of the whole bite is defined by the ascending slope. Only applies to `bite.type = "plat` (plateau-like bites.) Default: `10`.

slope.perc.end	Percentage of how much of the whole bite is defined by the descending slope. Only applies to bite.type = "plat (plateau-like bites.) Default: same as slope.perc.start.
jit	Jitter along the whole time series. Default: NULL.
bite.type	String: either "sin" or "plat" for sinusoidal or plateau-like bites, respectively. Default: "sin".
plot	Logical. If TRUE, the simulated time series will be plotted to the active plot device. Default: TRUE.

Value

Returns a tibble with the columns t and y containing simulated bites.

Examples

```
# simulate a time series with sinusoidal bites
# where the peaks are located in the centers of the bites.
simulate_bites(no.of.bites = 5,
               length.of.bite = 1000,
               length.of.series = 10000,
               max.y = 5,
               max.y.jit = 15,
               jit = 0.5,
               peak.pos = 0.5,
               bite.type = "sin",
               plot = TRUE)

# simulate a time series with sinusoidal bites
# where the peaks are located towards the ends of the bites.
simulate_bites(no.of.bites = 5,
               length.of.bite = 1000,
               length.of.series = 10000,
               max.y = 5,
               max.y.jit = 15,
               jit = 0.5,
               peak.pos = 0.8,
               bite.type = "sin",
               plot = TRUE)

# simulate a time series with plateau-like bites
simulate_bites(no.of.bites = 5,
               length.of.bite = 1000,
               length.of.series = 10000,
               max.y = 5,
               max.y.jit = 15,
               jit = 1,
               bite.type = "plat",
               plot = TRUE)

# simulate a time series with plateau-like bites
# with slowly ascending bite start and abprupt bite end.
```

```
simulate_bites(no.of.bites = 5,
               length.of.bite = 1000,
               length.of.series = 10000,
               max.y = 5,
               max.y.jit = 15,
               slope.perc.start = 60,
               slope.perc.end = 10,
               jit = 1,
               bite.type = "plat",
               plot = TRUE)
```

`sort_files` *Sorts files after corrections*

Description

The files of each of the various possible correction steps (cropping, amplififer correction, drift correction) are all located in their own folders. This function gets all files that represent the last correction step of a given measurement out of all those folders and saves them in the `results.folder`.

Usage

```
sort_files(data.folders, results.folder, move = FALSE)
```

Arguments

`data.folders`	Character vector containing full folder paths of folders to check. This list must be sorted according to the chronology of previous file editing. If a measurement exists in the last folder, this is copied or moved into the `results.folder`, and files of the same measurement located in the other folders will be ignored. Hence, the one file per measurement that underwent most correction steps will be stored in the `results.folder`, while the rest of the files of the same measurement remain in place.
`results.folder`	Character string defining the full path to the folder where the desired files will be stored.
`move`	A logical value specifying if files should be moved (`move = TRUE`) or copied (`move = FALSE`). Default: `FALSE`.

Details

The function will look for leading numbers in the file names specifying the measurement number to find corresponding files in the different folders. E.g., it will identify "0001_ABCD.csv", "0001_ABCD_ampdriftcorr.csv", and "0001_ABCD_ampdriftcorr_baselincorr.csv" as stemming from the same measurement and sort them accordingly.

Value

This functions does not create new files but sorts existing files. It does, however, create the `results.folder` in case it did not exist before.

Examples

```
# define data.folders
data.folders <- c("./raw",
                  "./cropped",
                  "./ampdriftcorr",
                  "./baselinecorr")

# define the folder in which one corrected file per original raw measurement
#   should be stored.
results.folder <- "./corrected"

# run the file sorting - commented out to pass package tests
# sort_files(data.folders = data.folders,
#               results.folder = results.folder,
#               move = FALSE)
```

`summarize_measurements` *Summarize Table*

Description

Finds minimum, maximum and standard deviation of force per measurement and taxon and creates summary tibble.

Usage

```
summarize_measurements(df, var1, var2)
```

Arguments

`df`	Data frame or tibble containing at least three columns. The column names must contain the grouping variables defined in `var1` and `var2` and the column `force` (time series of force measurements).
`var1`	A character string defining the column to calculate minimal and maximal force values per measurement. This must be the column that contains the unique measurement ID, e.g. measurement number.
`var2`	A character string defining the column for which the summary should be calculated.

Value

A tibble summarizing the input data frame `df`. The resulting tibble will contain the columns `t`, `force`, `measurement`, `species`, `specimen`, `amp`, `lever.ratio`, `max.F.measurement`, `mean.F.specimen`, `max.F.specimen`, `sdv.max.F.specimen`, `n.measurements.in.specimen`.

Examples

```
# Using the forceR::df.all.200.tax dataset:

# sumarize by measurement and specimen
df.summary.specimen <- summarize_measurements(df = df.all.200.tax,
                                              var1 = "measurement",
                                              var2 = "specimen")


#  plot results
## Not run:
require(ggplot2)
ggplot(data = df.summary.specimen, mapping = aes(x=specimen,y=max.F.measurement)) +
  geom_jitter(aes(color='blue'),alpha=0.7) +
  geom_boxplot(fill="bisque",color="black",alpha=0.3) +
  # scale_y_log10() +
  labs(y="max(F)/specimen") +
  guides(color="none") +
  theme_minimal()

## End(Not run)
```

---

today *Get Today's Date as String*

Description

Creates a character string containing today's date in the format "yyyy_mm_dd" which can be used in file names.

Usage

```
today()
```

Value

`date.string` a character string of today's date in the format "yyyy_mm_dd"

---

y_to_force *Convert Time Series to Force*

Description

Converts a time series, e.g. a continuous voltage measurement from a sensor to force data according to an amplification value and, depending on the measurement setup, the lever ratio of the rocker forwarding the force from the point the force acts on to the sensor.

Usage

```
y_to_force(df, classifier, measurement.col)
```

Arguments

`df`	Data frame or tibble in the below mentioned format. This should contain the time series, with one line per time step and measurement.
`classifier`	Classifier in the below mentioned format.
`measurement.col`	Character string. If `measurement.col` is not defined, the whole input data frames will be treated as if it was just one single time series. This is okay for data frames like that indeed only contain one time series, but for data frames with multiple time series, a grouping column needs to be defined. Default: `NULL`

Details

These values should be stored in a `classifier` (s. below). At the same, it adds `specimen` and `species` info from the respective columns of the `classifier`.

The `classifier` should have the following format:

species	specimen	measurement	amp	lever.ratio
species.1	specimen.1	measurement.1	amp.1	lever.ratio.1
...	...	...	...	...
species.n	specimen.n	measurement.n	amp.n	lever.ratio.n

It must contain one row per unique measurement number that is present in the df.

The force (`F`) in Newton is calculated *via* the following formula:

`F = y * lever.ratio * (1 / amp)`

where `y` is the measurement series, e.g. in `V`,
`amp` is the amplification value, e.g. in `V/N`,
and `lever.ratio` is the mechanical lever ratio of the measurement setup.

`df` should have the following format:

t	y	measurement
t.1	y.1	measurement.1
t.2	y.2	measurement.1
...	...	...
t.n	y.n	measurement.1
t.1	y.1	measurement.2
t.2	y.2	measurement.2

```
...   ...       ...
t.m   y.m   measurement.2
...   ...       ...
t.o   y.o   measurement.o
```

Value

Returns a tibble in the same format as the input tibble with an additional column called '"'force'.

Examples

```
# convert y column of df.all to force column and add taxonomic data
# using info from classifier
df.all.tax <- y_to_force(df = forceR::df.all.200,
                      classifier = forceR::classifier,
                      measurement.col = "measurement")

df.all.tax
```

Index