Package 'cycleRtools'

December 2, 2015

Title Tools for Cycling Data Analysis

Version 1.0.4

Description A suite of functions for analysing cycling data.

Depends R (>= 3.2.1)

SystemRequirements Java (>= 1.5)

License MIT + file LICENSE

LazyData true

LinkingTo Rcpp

Imports Rcpp, stats, graphics, grDevices, utils

Suggests xml2, raster, pspline, minpack.lm, changepoint, knitr, rmarkdown, RCurl, parallel

VignetteBuilder knitr

URL https://github.com/jmackie4/cycleRtools

RoxygenNote 5.0.1

NeedsCompilation yes

Author Jordan Mackie [aut, cre]

Maintainer Jordan Mackie <jmackie@protonmail.com>

Repository CRAN

Date/Publication 2015-12-02 00:01:51

R topics documented:

`convert_time` *Reformat time.*

Description

Functions perform interconversion between "HH:MM:SS" format and seconds.

Usage

```
convert_from_time(x)

convert_to_time(x)
```

Arguments

x	either a character string of the form "HH:MM:SS" ("HH" is optional) or a seconds value; can accept vectors.

Value

seconds value(s) for *from*, and "HH:MM:SS" character string(s) for *to*.

CP_model_inv *Inverse critical power modelling.*

Description

Generate critical power parameters via a linearised inverse time model. Mainly useful when the length of inputs is == 2, making `nls` inappropriate.

Usage

```
CP_model_inv(P, tsec)
```

Arguments

`P`	numeric; maximal mean power values for time periods given in the tsec argument.
`tsec`	numeric; time values (seconds) that correspond to elements in P.

Value

a named vector of parameters (CP and W').

cycling_data *Example cycling data.*

Description

Formatted cycling data from a Garmin head-unit. imported via `cycleRtools::read_fit("file_path", format = TRUE)`.

Usage

```
cycling_data
```

Format

An object of class c("cycleRdata", "data.frame").

timer.s an ongoing timer (seconds). Stoppages are not recorded per se, but rather represented as breaks in the continuity of the timer.

timer.min as above, but in units of minutes.

timestamp "POSIXct" values, describing the actual time of day.

delta.t delta time values (seconds). Obtained via base::diff(timer.s).

lat latitude values (units: degrees).

lng longitude values (units: degrees).

distance.km cumulative distance (kilometres).

speed.kmh speed in kilometres per hour.

elevation.m altitude in metres.

delta.elev delta elevation (metres). Obtained via base::diff(elevation.m).

VAM "vertical ascent metres per second".

power.W power readings (watts).

power.smooth.W an exponentially-weighted 25-second moving average of power.W values.

work.J cumulative work (joules).

Wexp.kJ W' expended in units of kilojoules. See ?Wbal and references therein.

cadence.rpm pedalling cadence in units of revolutions per minute (rpm).

lap a numeric vector of lap "levels". Will only have values > 1 if lap data is available.

.elevation.corrected.m added for the sake of example; see package vignette.

`diff_section` *Section data according to breaks.*

Description

Generates a vector of "section" values/levels according to differences in the supplied vector. The function simply rolls over `x`, incrementing the return vector every time there is a significant break in the pattern of differences between adjacent elements of `x`. In practical terms, if `x` is a series of timestamp values (see example), every time there is a significant break in the timer (> 10 sec currently), the return vector is incremented by 1.

Usage

```
diff_section(x)
```

Arguments

`x`	a numeric vector (e.g. a timer column) that increments uniformly. When there is a **significant** break in this uniformity, a new section is created, and so forth.

Value

a vector of the same shape as x.

Examples

```
t_sec <- c(1:10, 40:60, 100:150)       # Discontinuous timer values.
pwr   <- runif(length(t_sec), 0, 400)  # Some power values.
x     <- data.frame(t_sec, pwr)
# Generate section levels.
x$section <- diff_section(x$t_sec)
print(x)
```

download_elev_data *Download geographical elevation data.*

Description

Downloads elevation data files to the working directory for use with `elevation_correct`. Requires package `raster` to be installed.

Usage

```
download_elev_data(country = "all")
```

Arguments

`country`	character string; the ISO3 country code (see `raster::getData("ISO3")`) for which to download the data. If "all", then all available data is downloaded - this may take some time.

Value

nothing.

See Also

`elevation_correct`.

elevation_correct *Generate reliable elevation data.*

Description

Using the latitude and longitude columns of the supplied *formatted* data, a vector of elevation values is returned of the same length. If no elevation data files exist within the working directory, files are first downloaded. Note that NAs in the data will return corresponding NAs in the corrected elevation.

Usage

```
elevation_correct(data, country)
```

Arguments

`data`	a dataset with longitude ("lng") and lattitude ("lat") columns.
`country`	character string; the country to which the data pertain, given as an ISO3 code (see `raster::getData("ISO3")`)

Value

a vector of elevation values. If there is an error at any stage, a vector of NAs is returned.

See Also

download_elev_data.

Examples

```
## Not run:
data(cycling_data)
cycling_data$elevation.m <- elevation_correct(cycling_data, "GBR")

## End(Not run)
```

GC *GoldenCheetah (>v3.3) interface.*

Description

Functions for interfacing R with GoldenCheetah. Requires the RCurl package to be installed.

Usage

```
GC_activity(athlete.name, activity, port = 12021, format = TRUE)

GC_metrics(athlete.name, date.rng = NULL, port = 12021)

GC_mmvs(type = "watts", date.rng = NULL, port = 12021)
```

Arguments

athlete.name	character; athlete of interest in the GoldenCheetah data directory. Typically of the form "First Last".
activity	character; file path to a GoldenCheetah activity(.json) file. Typically located in "~/.goldencheetah/Athlete Name/activities/".
port	http server port number. 12021 unless deliberatley changed in the httpserver.ini file.
format	format activity data to an object of class "cycleRdata". Ensures compatibility with other functions in this package – see read.
date.rng	a vector of length two that can be converted to an object of class "Date" via as.Date. Must be specified for GC_mmvs; optional for GC_metrics.
type	the type of maximal mean values to return. See details.

Details

As of GoldenCheetah (GC) version 3.3, the application is ran with a background http server to ease integration with external analysis software (such as R). When an instance of GoldenCheetah is running, or the application is initiated from the command line with the '–server' option, these functions can be used to interface with athlete data. Relevant documentation can be found here.

`GC_activity` behaves similarly to `read` functions in this package, importing data from saved GC .json files.

`GC_metrics` returns summary metrics for either: all available rides if `date.rng = NULL`; or rides within a specified date range if dates are given.

`GC_mmvs` retuns best maximal mean values for data specified in the `type` argument. Possible options for `type` are: "watts", "hr", "cad", "speed", "nm", "vam", "xPower", or "NP". See also `mmv`.

`interval_data` *Example cycling interval data.*

Description

Formatted cycling data from a Garmin head-unit. imported via `cycleRtools::read_fit("file_path", format = TRUE)`. Included to demonstrate the use of `interval_detect()`.

Usage

```
interval_data
```

Format

An object of class c("cycleRdata", "data.frame").

timer.s an ongoing timer (seconds). Stoppages are not recorded per se, but rather represented as breaks in the continuity of the timer.

timer.min as above, but in units of minutes.

timestamp "POSIXct" values, describing the actual time of day.

delta.t delta time values (seconds). Obtained via base::diff(timer.s).

lat latitude values (units: degrees).

lng longitude values (units: degrees).

distance.km cumulative distance (kilometres).

speed.kmh speed in kilometres per hour.

elevation.m altitude in metres.

delta.elev delta elevation (metres). Obtained via base::diff(elevation.m).

VAM "vertical ascent metres per second".

power.W power readings (watts).

power.smooth.W an exponentially-weighted 25-second moving average of power.W values.

work.J cumulative work (joules).

Wexp.kJ W' expended in units of kilojoules. See ?Wbal and references therein.

lap a numeric vector of lap "levels". Will only have values > 1 if lap data is available.

`interval_detect` *Detect Intervals in a Ride.*

Description

Often a ride will contain intervals/efforts that are not in any way marked in the device data (e.g. as "laps"). Using changepoint analysis, it is possible to retrospectively identify these efforts. This is contingent on supplying the number of changepoints to the underlying algorithm, simplified here as a `sections` argument. For example, if there are two efforts amidst a ride, this means we are looking to identify 5 *sections* (i.e. neutral-effort-neutral-effort-neutral). See **Examples**. Requires package `"changepoint"`.

Usage

```
interval_detect(data, sections, plot = FALSE, ...)
```

Arguments

`data`	a **formatted** dataset produced by `read*()`.
`sections`	how many sections should be identified? see **Description**.
`plot`	logical; if TRUE, graphically displays the resultant sections.
`...`	graphical parameters to be passed to `par()`. Not used if `plot = FALSE`.

Value

if `plot = TRUE` nothing is returned. If `plot = FALSE` (default) a vector of section "levels" is returned.

Examples

```
# "interval_data" is a ride data set, which involved two 10 minute efforts
# during an otherwise steady ride.
data(interval_data)
interval_data$interval <- interval_detect(interval_data, 5, FALSE)
# Two efforts = 5 sections.
# Were the two 10 minute efforts properly identified?
tapply(interval_data$delta.t, interval_data$interval, sum) / 60
# Plot
interval_detect(interval_data, 5, TRUE)
```

LT *Lactate Thresholds*

Description

Model lactate threshold markers from work rate (power) and blood lactate values. Requires package "pspline".

Usage

```
LT(WR, La, sig_rise = 1.5, plots = TRUE)
```

Arguments

WR	a numeric vector of work rate values. Typically these would be the work rates associated with stages in an incremental exercise test.
La	a numeric vector of blood lactate values (mmol/L) associated with the stages described in WR.
sig_rise	numeric; a rise in blood [Lactate] that is deemed significant. Default is 1.5 mmol/L.
plots	should outputs be plotted?

Details

This function is a slightly modified version of that written by Newell *et al.* (2007) and published in the Journal of Sport Sciences (see references). The original source code, which also includes other functions for lactate analysis, can be found here.

Value

a data frame of model outputs, and optionally a matrix of plots.

References

John Newell , David Higgins , Niall Madden , James Cruickshank , Jochen Einbeck , Kenny McMillan & Roddy McDonald (2007) Software for calculating blood lactate endurance markers, Journal of Sports Sciences, 25:12, 1403-1409, DOI.

See Also

Newell *et al.*'s Shiny app.

Examples

```
# This data is included with Newell et al's source code.
WR <- c(50, 75, 100, 125, 150, 175, 200, 225, 250)
La <- c(2.8, 2.4, 2.4, 2.9, 3.1, 4.0, 5.8, 9.3, 12.2)
LT(WR, La, 1.5, TRUE)
```

mmv *Maximal mean values.*

Description

Calculate maximal mean values for specified time periods.

Usage

```
mmv(data, column, windows, delta = NULL, verbose = TRUE, .uniform = FALSE,
  .string = FALSE)
```

Arguments

data	a **formatted** dataset produced by read*().
column	column in data giving the values of interest. Needn't be quoted.
windows	window size(s) for which to generate best averages, given in seconds.
delta	the incremement with which to initially make the data uniform. If NULL (default), the most appropriate value is estimated.
verbose	should messages be printed to the console?
.uniform	are the rows of the data already uniform?
.string	are column name arguments given as character strings? A backdoor around non-standard evaluation. Mainly for internal use.

Value

a matrix object with two rows: 1) best mean value(s) and 2) the time at which that value was recorded

See Also

For a more generic and efficient version of this function, see mmv2

Examples

```
data(cycling_data)
# Generate best powers for 5, 10 and 20 minutes.
t_sec <- c(5, 10, 20) * 60
x <- mmv(cycling_data, power.W, t_sec)
# Show when those values were recorded in minutes
x[2, ] / 60
```

mmv2 *Efficient maximal mean values.*

Description

A more efficient implementation of mmv. Simply takes a vector (x) of values and rolls over them element wise by defined windows. Returns a vector of maximum mean values for each window size.

Usage

```
mmv2(x, windows)
```

Arguments

x	a numeric vector of values.
windows	window size(s) (in element units) for which to generate maximum mean values.

Value

a vector of length(windows).

NP *Normalised power.*

Description

Calculate a Normalised Power value. Normalised Power is a registered trademark of Peaksware Inc.

Usage

```
NP(data, delta = NULL, verbose = TRUE)
```

Arguments

data	a **formatted** dataset produced by read*().
delta	optional; the sampling frequency of data (in seconds per sample).
verbose	should messages be printed to the console?

Value

a Normalised Power value.

Examples

```
data(cycling_data)
NP(cycling_data)
```

`predict.Ptmodels` *Predict Power or Time*

Description

Given a Ptmodels `object`, the predict.Ptmodels will produce a named numeric vector of either time (seconds) or power (watts) values according to the `x` and `y` arguments

Usage

```
## S3 method for class 'Ptmodels'
predict(object, x, y = "P", ...)
```

Arguments

`object`	an object of class "Ptmodels".
`x`	the value for which to make a prediction, see below.
`y`	the type of variable to predict: "P" will produce a power prediction, and hence assume `x` is a time value; and "tsec" vice versa.
`...`	further arguments passed to or from other methods.

Value

a named numeric vector of predicted values. Names correspond to their respective models.

`Pt_model` *Power-time modelling.*

Description

Model the Power-time (Pt) relationship for a set of data. This is done via nonlinear least squares regression of four models: an inverse model; an exponential model; a bivariate power function model; and a three parameter inverse model. An S3 object of class "Ptmodels" is returned, which currently has methods for print, coef, summary, and predict. If inputs do not conform well to the models, a warning message is generated. This function can make use of `minpack.lm::nlsLM`.

Usage

```
Pt_model(P, tsec)
```

Arguments

P	a numeric vector of maximal mean power values for time periods given in the tsec argument.
tsec	a numeric vector of time values that (positionally) correspond to elements in P.

Value

returns an S3 object of class "Ptmodels".

References

R. Hugh Morton (1996) A 3-parameter critical power model, Ergonomics, 39:4, 611-619, DOI.

Examples

```
data(Pt_prof)
P    <- unname(Pt_prof)
tsec <- as.numeric(names(Pt_prof))
# Generate model object
m <- Pt_model(P, tsec)
# View
print(m)
# Plot
plot(P ~ tsec, cex = 0.2)
with(m$Pfn, curve(inv(x), add = TRUE, col = "red"))
with(m$Pfn, curve(exp(x), add = TRUE, col = "blue"))
with(m$Pfn, curve(pwr(x), add = TRUE, col = "purple"))
with(m$Pfn, curve(thrp(x), add = TRUE, col = "green"))
legend("topright", legend = round(m$table$RSE, 2),
       text.col = c("red", "blue", "purple", "green"),
       title = "RSE",title.col = "black",
       bty = "n")
```

Pt_prof *An example Power-time profile*

Description

A named numeric vector of best power values, where names are the corresponding time periods (seconds).

Usage

```
Pt_prof
```

Format

An object of class array of length 358.

pwr_TRIMP *Power-Based TRaining IMPulse.*

Description

Calculate a *normalised* TRIMP value using power data. This is a power-based adaptation of Banister's TRIMP, whereby critical power (CP) is assumed to represent 90 normalised to the score associated with one-hour's riding at CP, to aid interpretation.

Usage

```
pwr_TRIMP(data, CP)
```

Arguments

data	a **formatted** dataset produced by read*().
CP	a critical power value (watts).

Value

a normalised TRIMP score.

References

Morton, R.H., Fitz-Clarke, J.R., Banister, E.W., 1990. Modeling human performance in running. Journal of Applied Physiology 69, 1171-1177.

read *Read cycling device data.*

Description

Read data from a cycling head unit into the R environment; optionally formatting it for use with other functions in this package.

Usage

```
read(file = file.choose(), format = TRUE)

read_fit(file = file.choose(), format = TRUE)

read_pwx(file = file.choose(), format = TRUE)

read_srm(file = file.choose(), format = TRUE)

read_tcx(file = file.choose(), format = TRUE)
```

Arguments

file	character; path to the file.
format	logical; should data be formatted?

Details

Returns a data frame with all data parsed from the file; with an additional lap (factor) column appended in the case of read_fit().

Note that most functions within this package depend on imported data being formatted; i.e. read*("file_path", format = Hence, unless the raw data is of particular interest and/or the user wants to process it manually, the format argument should be TRUE (default). When working with a formatted dataset, do not change existing column names. The formatted data structure is described in detail in cycling_data.

Garmin .fit file data is parsed with the java command line tool provided in the FIT SDK. The latest source code and licensing information can be found at the previous link.

SRM device files (.srm) are also parsed at the command line, provided Rainer Clasen's srmio library is installed and available. The associated GitHub repo' can be found here.

Value

a data frame object.

Functions

- read: A wrapper for read_* functions that chooses the appropriate function based on file extension.
- read_fit: Read a Garmin (Ltd) device .fit file. This invokes system2 to execute the FitCSVTool.jar command line tool (see FIT SDK). Hence, this function requires that Java (JRE/JDK) binaries be on the system path.
- read_pwx: Read a Training Peaks .pwx file. Requires the "xml2" package to be installed. Will make use of the "parallel" package if available.
- read_srm: Read an SRM (.srm) file. This requires Rainer Clasen's srmio library to be installed and on the system path.
- read_tcx: Read a Garmin .tcx file. Requires the "xml2" package to be installed. Will make use of the "parallel" package if available.

---

reset *Reset a dataset or vector.*

Description

Subtracts the first element from all other elements.

Usage

```
reset(x)
```

Arguments

x	a numeric vector or formatted cycling dataset (i.e. class "cycleRdata").

Details

if x is a formatted dataset from a read* function, all the columns are reset as appropriate. This can be useful after subsetting a ride dataset, for example. Otherwise, this is a simple shortcut for x - x[[1]].

Value

either a data frame or vector, depending on the class of x.

Examples

```
data(cycling_data)
# Remove first minute of data and reset.
data_raw <- cycling_data[cycling_data$timer.s > 60, ]
data_reset <- reset(data_raw)
# Compare...
data_raw$distance.km[[1]]
data_reset$distance.km[[1]]
```

---

ride_time *Calculate ride time.*

Description

A simple function for calculating ride time as opposed to elapsed time.

Usage

```
ride_time(x, delta = NULL)
```

Arguments

x	a vector of time values.
delta	numeric; the typical interval between time values, if NULL a best estimate is used.

Value

ride time in the same units as x.

Examples

```
t_sec <- c(1:20, 50:70)
# elapsed time
max(t_sec)
# ride time.
ride_time(t_sec)
```

`rollmean_smth` *Smooth a column of a dataset.*

Description

Smooth data with a right-aligned (zero-padded) rolling average. This is intended for time-based smoothing - e.g. a 30-second moving average. Hence, as rolling operations are performed row-wise in in the interest of efficiency, the data must first be made time-uniform by row. Hence, this is done via `uniform` by the first column of the data; thus allowing the `column` to be smoothed.

Usage

```
rollmean_smth(data, column, smth.pd, ema = FALSE, delta = NULL,
  verbose = TRUE, .uniform = FALSE, .string = FALSE)
```

Arguments

`data`	a data.frame or matrix object to use for smoothing. The data is made `uniform` on the basis of the first column.
`column`	the column name of the data to be smoothed, needn't be quoted.
`smth.pd`	numeric; the time period over which to smooth (seconds).
`ema`	should the moving average be exponentially weighted?
`delta`	the incremement with which to initially make the data uniform. If NULL (default) a best estimate is used.
`verbose`	should messages be printed to the console?
`.uniform`	are the rows of the data already uniform?
`.string`	are column name arguments given as character strings? A backdoor around non-standard evaluation. Mainly for internal use.

Value

a vector of the same length as the `column`.

Examples

```
## Not run:
data(cycling_data)
# Smooth power data with a 30 second moving average.
rollmean_smth(cycling_data, power.W, 30)
# Or alternatively, use an exponentially weighted moving average.
rollmean_smth(cycling_data, power.W, 30, ema = TRUE)

## End(Not run)
```

`smth_plot` *Smoothed data plot.*

Description

Create a plot with both raw and smoothed data lines.

Usage

```
smth_plot(data, x = "timer.s", yraw = "power.W", ysmth = "power.smooth.W",
  colour = "lap", ..., .string = FALSE)
```

Arguments

`data`	the dataset to be used.
`x`	column identifier for the x axis data.
`yraw`	column identifier for the (underlying) raw data.
`ysmth`	column identifier for the smoothed data.
`colour`	level identifier in `data` by which to colour lines. Or a colour name. Or simply a nonexistent column name (equivalent to "black").
`...`	further arguments to be passed to `plot()`.
`.string`	are column name arguments given as character strings? A backdoor around non-standard evaluation. Mainly for internal use.

Examples

```
data(cycling_data)
# Create some artificial laps.
cycling_data$lap <- floor(seq(from = 1, to = 5, along.with = cycling_data[, 1]))
# Plot:
smth_plot(cycling_data, timer.s, power.W, power.smooth.W, colour = "lap")
# Plot with a single blue line:
smth_plot(cycling_data, colour = "blue")  # Default arguments.
```

summary.cycleRdata *Summary method for cycleRdata class.*

Description

Relevant summary metrics for cycling data (class 'cycleRdata').

Usage

```
## S3 method for class 'cycleRdata'
summary(object, sRPE = NULL, CP = NULL,
  round_digits = NULL, prettynames = FALSE, .smoothpwr = "power.smooth.W",
  ...)
```

Arguments

object	object for which a summary is desired.
sRPE	optional; session Rating of Percieved Exertion (value between 1 and 10).
CP	optional; Critical Power value (Watts).
round_digits	optional; number of digits to round all metrics to.
prettynames	self explanatory.
.smoothpwr	character string; column name of smoothed power values. For internal use.
...	further arguments passed to or from other methods.

Value

a list object.

Examples

```
data(cycling_data)
summary(cycling_data)
```

TSS *Training stress score.*

Description

Calculate a Training Stress Score (TSS). TSS is a registered trademark of Peaksware Inc.

Usage

```
TSS(data, threshold, verbose = TRUE)
```

Arguments

data	a **formatted** dataset produced by read*().
threshold	a threshold value - e.g. FTP.
verbose	should messages be printed to the console?

Value

a TSS score.

---

uniform *Create a uniform data.frame.*

Description

Returns a data frame with rows that are uniformly incremented along the first column. Useful for rolling operations, whereby a rolling time window is of interest.

Usage

```
uniform(data, delta = NULL, verbose = TRUE, .return_delta = FALSE)
```

Arguments

data	a data.frame or matrix object to make uniform. This is done on the basis of the first column.
delta	the incremement by which to make the data uniform. If NULL (default) the most appropriate value is estimated.
verbose	should messages be printed to the console?
.return_delta	if TRUE the delta value is assigned to the calling environment. Intended for internal use.

Value

A uniform data frame. An index column is appended so that original data values can be extracted - e.g. after a rolling operation.

See Also

source code for rollmean_smth

Examples

```
t_sec <- c(1:10, 20:40, 50)           # Discontinuous timer values.
pwr   <- runif(length(t_sec), 0, 400)  # Some power values.
x     <- data.frame(t_sec, pwr)
uniform(x)
```

`Wbal` *W' balance.*

Description

Generate a vector of W' balance values from time and power data. The underlying algorithm is published in Skiba *et al.* (2012).

Usage

```
Wbal(data, time = "timer.s", pwr = "power.W", CP, .string = FALSE)
```

Arguments

`data`	a data.frame/matrix object with time and power columns.
`time`	character; name of the time (seconds) column in `data`.
`pwr`	character; name of the power (watts) column in `data`.
`CP`	a critical power value for use in the calculation.
`.string`	are column name arguments given as character strings? A backdoor around non-standard evaluation. Mainly for internal use.

Details

The algorithm used here, while based on Dr Phil Skiba's model, differs in that values are positive as opposed to negative. The original published model expressed W' balance as W' minus W' expended, the latter recovering with an exponential time course when P < CP. An issue with this approach is that an athlete might be seen to go into negative W' balance. Hence, to avoid assumptions regarding available W', this algorithm returns W' expended (and its recovery) as positive values; i.e. a ride is begun at 0 W' expended, and it will *increase* in response to supra-CP efforts.

Value

A numeric vector of W' balance values, in **kilojoules**.

References

Skiba, P. F., W. Chidnok, A. Vanhatalo, and A. M. Jones. Modeling the Expenditure and Reconstitution of Work Capacity above Critical Power. Med. Sci. Sports Exerc., Vol. 44, No. 8, pp. 1526-1532, 2012. PubMed link.

See Also

`Wbal_plots`.

`Wbal_plots` *W' balance plots.*

Description

Generate three plots that effectively summarise a whole cycling dataset.

Usage

```
Wbal_plots(data, x = 1, n = c(1, 2, 3), xlab = NULL, xlim = NULL,
  CP = NULL, laps = FALSE, ...)
```

Arguments

`data`	a **formatted** dataset produced by `read*()`.
`x`	numeric; `1` will plot against time (sec); `2` will plot against time (minutes); and `3` will plot against distance.
`n`	plots to be created (see details).
`xlab`	character; x axis label for bottom plot.
`xlim`	given in terms of `x`.
`CP`	a value for critical power annotation.
`laps`	logical; should laps be seperately coloured?
`...`	graphical parameters to be passed to `par()`.

Details

The `n` argument describes plot options such that:

1. plots W' balance (kJ).
2. plots power data (W).
3. plots an elevation profile (m).

These options can be concatenated to produce a stack of plots as desired.

Value

a variable number of plots.

See Also

`Wbal`.

Examples

```
data(cycling_data)
Wbal_plots(cycling_data, x = 2, n = c(1, 2, 3), CP = 300)
# Show just W' balance.
Wbal_plots(cycling_data, x = 2, n = 1)
# Elevation profile on top
Wbal_plots(cycling_data, x = 2, n = c(3, 1))
# Zoom all plots to 20-40 km.
Wbal_plots(cycling_data, x = 3, CP = 300, xlim = c(20, 40), xaxs = "i")
```

---

zdist_plot *Zone-time distribution plot.*

Description

Display the time distribution of values within a dataset. The distribution can also be partitioned into zones if the zbounds argument is not NULL.

Usage

```
zdist_plot(data, column = "power.W", binwidth = 10, zbounds = NULL,
  .string = FALSE, ...)
```

Arguments

data	a **formatted** dataset produced by read*().
column	column in data giving the values of interest. Needn't be quoted.
binwidth	how should values in column be binned? E.g. bindiwdth = 10 will create 10 watt bins if column is power data.
zbounds	optional; a numeric vector of zone boundaries.
.string	are column name arguments given as character strings? A backdoor around non-standard evaluation. Mainly for internal use.
...	arguments to be passed to barplot() and/or graphical parameters.

Value

nothing; a plot is sent to the current graphics device.

Examples

```
data(cycling_data)
zdist_plot(
 data = cycling_data,
 column = power.W,
 binwidth = 10,
 zbounds = c(100, 200, 300),
```

```
 xlim = c(110, 500)
)
```

---

zone_index *Index zones.*

Description

Generate a vector of zone "levels" from an input vector and defined boundaries.

Usage

```
zone_index(x, zbounds)
```

Arguments

x	numeric; values to be "zoned".
zbounds	numeric; values for zone boundaries.

Value

a numeric vector of zone values of the same length as `x`. The number of zone levels will be `length(zbounds) + 1`.

Examples

```
data(cycling_data)
cycling_data$zone <- zone_index(cycling_data$power.W, c(100, 200, 300))
```

---

zone_time *Calculate time in zones.*

Description

Given a vector of zone boundaries, sums the time spent in each zone.

Usage

```
zone_time(data, column = "power.W", zbounds, pct = FALSE, .string = FALSE)
```

Arguments

`data`	a **formatted** dataset produced by `read*()`.
`column`	the column name of the data to which the zone boundaries relate.
`zbounds`	numeric; zone boundaries.
`pct`	should percentage values be returned?
`.string`	are column name arguments given as character strings? A backdoor around non-standard evaluation. Mainly for internal use.

Value

a data frame of zone times.

Index