Package ‘cycleRtools’

January 18, 2016

Title Tools for Cycling Data Analysis
Version 1.1.1
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, xml2, stats, graphics, grDevices, utils
Suggests changepoint, minpack.lm, RCurl, raster, pspline, leaflet,
        knitr, rmarkdown
VignetteBuilder knitr
URL https://github.com/jmackie/cycleRtools
RoxygenNote 5.0.1
NeedsCompilation yes
Author Jordan Mackie [aut, cre]
Maintainer Jordan Mackie <jmackie@protonmail.com>
Repository CRAN
Date/Publication 2016-01-18 17:29:02

R topics documented:

correct_time ................................................................. 2
cycleRdata ................................................................. 3
diff_section ............................................................... 4
download_elev_data ...................................................... 5
elevation_correct .......................................................... 5
GC ............................................................................. 6
interval_detect .............................................................. 7
LT .............................................................................. 9
**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 numeric seconds values.

**Value**

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

Examples

```r
x <- c("00:21:05", "25:51", NA, "00:26:01.1", "01:05:02.0")
x <- convert_from_time(x)
print(x)
x <- convert_to_time(x)
print(x)
```

cycleRdata  cycleRdata class

Description

A class for imported ride files intended to ease integration with package functionality. Produced by invoking `read_ride` (or equivalent) with the argument `format = TRUE`. Fundamentally, cycleRdata objects are a special type of `data.frame`; special in the sense that column names are predefined and assumed to be present in the class’ associated methods. Modification of these column names will lead to errors. See below for a description of the format.

Usage

```r
is.cycleRdata(x)
as.cycleRdata(x)
```

Arguments

- `x` an object to be tested/coerced.

Format

The columns of cycleRdata objects are structured as such:

- `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).
- `lat` latitude values (degrees).
- `lng` longitude values (degrees).
- `distance.km` cumulative distance (kilometres).
- `speed.kmh` speed in kilometres per hour.
- `elevation.m` altitude in metres.
- `delta.elev` delta elevation (metres).
- `VAM` "vertical ascent metres per second".
power.W  power readings (Watts).

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

work.kJ  cumulative work (kilojoules).

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

cadence.rpm  pedalling cadence (revolutions per minute).

hr.bpm  Heart rate (beats per minute).

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

---

**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 (stop argument) 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 (e.g. >10 sec), the return vector is incremented by 1.

**Usage**

\[
\text{diff_section}(x, \text{br})
\]

**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.
- **br**  criterion for a significant break in terms of \( x \).

**Value**

a vector of the same length as \( x \).

**Examples**

```r
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, br = 10)  # 10 second breaks.
print(x)
split(x, x$section)

# Using "intervaldata", which has a large stop.
data(intervaldata)
```
download_elev_data

intervaldata$section <- diff_section(intervaldata$timer.s, br = 20)
sp <- split(intervaldata, intervaldata$section)

# Plot.
eplot <- function(x) cyclertools:::elev_plot(x, "timer.min")
layout(matrix(c(1, 2, 1, 3), 2, 2))
eplot(cyclertools:::expand_stops(intervaldata))
eplot(sp[[1]])
eplot(sp[[2]])

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, files are downloaded to the working directory.

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 latitude ("lon") 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(ridedata)

## When run the first time, geographical data will need to be downloaded.
ridedata$elevation.corrected <- elevation_correct(ridedata, "GBR")

## A Bland-Altman-type plot.
difference <- ridedata$elevation.m - ridedata$elevation.corrected
plot(difference ~ ridedata$timer.min, cex = 0.2, ylab = "raw minus corrected")
m <- mean(difference, na.rm = TRUE); stdev <- sd(difference, na.rm = TRUE)
abline(h = c(m + c(-stdev, 0, stdev)), lty = c(1, 2, 1), col = "red")

## 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_ride.

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 restful web service api 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_ride 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 returns 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.

Description

Section a ride file according to power output.

Usage

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

Arguments

data a formatted dataset produced by read*().

sections how many sections should be identified? Includes stoppages.

plot logical; if TRUE, graphically displays the resultant sections.

... graphical parameters to be passed to par(). Ignored if plot = FALSE.
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.

Depends on the package "changepoint".

Value

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

Examples

data(intervaldata)

## "intervaldata" is a ride that includes two efforts (2 & 5 minutes) and a cafe stop. The efforts are marked in the lap column, which we can use as a criterion.

with(intervaldata, tapply(X = delta.t, INDEX = lap, sum)) / 60  # Minutes.

## The above shows the efforts were laps two and four. What was the power?
with(intervaldata, tapply(X = power.W, INDEX = lap, mean))[c(2, 4)]

## And for the sake of example, some other summary metrics...
l <- split(intervaldata, intervaldata$lap)
names(l) <- paste("Lap", names(l))  # Pretty names.
vapply(l, FUN.VALUE = numeric(3), FUN = function(x)
  c(t.min = ride_time(x$timer.s) / 60, NP = NP(x), TSS = TSS(x)))

## Could we have gotten the same information without the lap column?
## Two efforts and a cafe stop == 7 sections.
interval_detect(intervaldata, sections = 7, plot = TRUE)

## An overzealous start to the first effort is being treated as a separate section,
## so let's allow for an extra section...
interval_detect(intervaldata, sections = 8, plot = TRUE)

## Looks okay, so save the output and combine the second and third sections.
intervaldata$intv <- interval_detect(intervaldata, sections = 8, plot = FALSE)
intervaldata$intv[intervaldata$intv == 3] <- 2

## Are the timings as expected?
with(intervaldata, tapply(X = delta.t, INDEX = intv, sum)) / 60  # Minutes.

## Close enough!
i <- split(intervaldata, intervaldata$intv)
names(i) <- paste("Interval", seq_along(i))  # Pretty names.
toplot <- vapply(1, FUN.VALUE = numeric(3), FUN = function(x)
  c(t.min = ride_time(x$timer.s) / 60, NP = NP(x), TSS = TSS(x)))

print(toplot)

par(mfrow = c(3, 1))
mapply(function(r, ylab) barplot(
  toplot[r, c(1:3, 5:7)], names.arg = seq_along(toplot[r, c(1:3, 5:7)]),
  xlab = "Section", ylab = ylab),
  r = 1:3, ylab = c("Ride time (minutes)", "NP", "TSS"))

---

**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


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, deltat = NULL, character.only = 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.
deltat the sampling frequency of data in seconds per sample; typically 0.5 or 1. If NULL, this is estimated.
character.only 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 values and 2) the time at which those values were recorded

See Also

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

```r
data(ridedata)

## Best power for 5 and 20 minutes.
tsec <- c(5, 20) * 60
mmv(ridedata, power.W, tsec)

## Generate a simple critical power estimate.
tsec <- 2:20 * 60
pwrs <- mmv(ridedata, power.W, tsec)
m <- lm(pwrs[1, ] ~ 1 / tsec) # Simple inverse model.
coef(m)[1] # Intercept = critical power.

## More complex models...
m <- Pt_model(pwrs[1, ], tsec)
print(m)
## Extract the asymptote of the exponential model.
coef(m)$exp["CP"]
```

---

**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 `windows`. Returns a vector of maximum mean values for each window. NAs are not ignored.

**Usage**

```r
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)`.

**Examples**

```r
x <- rnorm(100, 500, 200)
mmv2(x, windows = c(5, 10, 20))
```
plot.cycleRdata  

Plot cycling data.

Description

Generate plots to effectively summarise a cycling dataset.

Usage

```r
## S3 method for class 'cycleRdata'
plot(x, y = 1:3, xvar = "timer.s", xlab = NULL,
     xlim = NULL, CP = attr(x, "CP"), laps = FALSE, breaks = TRUE, ...)
```

Arguments

- `x`: a "cycleRdata" object produced by `read*()`.
- `y`: numeric; plots to be created (see details).
- `xvar`: character; name of the column to be plotted as the `x`variable.
- `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 separately coloured?
- `breaks`: logical; should plot lines be broken when stationary? Will only show when `xvar` represents time values.
- `...`: graphical parameters, and/or arguments to be passed to or from other methods.

Details

The `y` 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 combined to produce a stack of plots as desired.

Value

a variable number of plots.
predict.Ptmodels

Examples

```r
## Not run:
data(riededata)

plot(riededata, xvar = "timer.min")
plot(riededata, xvar = "distance.km")

## With only two plots.
plot(riededata, y = c(2, 1))

## Using xlim, note that title metrics adjust.
plot(riededata, xvar = "timer.min", xlim = c(100, 150))

## Lap colouring.
data(intervaldata)
plot(intervaldata, xvar = "timer.min", laps = TRUE)

## End(Not run)
```

---

**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**

```r
## S3 method for class 'Ptmodels'
predict(object, x, xtype = c("pwr", "time"), ...)
```

**Arguments**

- **object**: an object of class "Ptmodels".
- **x**: the value for which to make a prediction.
- **xtype**: what is x? A power or a time value?
- **...**: further arguments passed to or from other methods.

**Value**

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

data(Pt_prof)  # Example power-time profile.

P  <- Pt_prof$pwr
tsec <- Pt_prof$time

mdls <- Pt_model(P, tsec)  ## Model.
print(mdls)

## What is the best predicted 20 minute power?
predict(mdls, x = 60 * 20, xtype = "time")

## How sustainable is 500 Watts?
predict(mdls, x = 500, xtype = "P") / 60  # Minutes.

## Create some plots of the models.
par(mfrow = c(2, 2), mar = c(3.1, 3.1, 1.1, 1.1))
plotargs <- alist(x = tsec, y = P, cex = 0.2, ann = FALSE, bty = "l")
mapply(function(f, m) {
  do.call(plot, plotargs)
  curve(f(x), col = "red", add = TRUE)
  title(main = paste0(rownames(m), "; RSE = ", round(m$RSE, 2)))
  legend("topleft", legend = m$formula, bty = "n")
  return()
}, f = mdls$pfn, m = split(mdls$table, seq_len(nrow(mdls$table))))

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 will make use of minpack.lm::nlsLM if available.

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


See Also

predict.Ptmodels

Examples

data(Pt_prof) # Example power-time profile.

P <- Pt_prof$pwr
tsec <- Pt_prof$time

mdls <- Pt_model(P, tsec) # Model.
print(mdls)

coef(mdls)
summary(mdls)
Description

Read data from a cycling head unit into the R environment; optionally formatting it for use with other functions in this package. Critical power and session RPE metrics can also be associated with the data and used by other functions (e.g. summary.cycleRdata).

Usage

read_ride(file = file.choose(), format = TRUE, CP = NULL, sRPE = NULL)
read_fit(file = file.choose(), format = TRUE, CP = NULL, sRPE = NULL)
read_pwx(file = file.choose(), format = TRUE, CP = NULL, sRPE = NULL)
read_srm(file = file.choose(), format = TRUE, CP = NULL, sRPE = NULL)
read_tcx(file = file.choose(), format = TRUE, CP = NULL, sRPE = NULL)

Arguments

file character; path to the file.
format logical; should data be formatted?
CP, sRPE optional; critical power and session RPE values to be associated with the data. Ignored if format = FALSE.

Details

Note that most functions within this package depend on imported data being formatted; i.e. read*("file_path", format = TRUE). 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 ridedata.

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_ride**: 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 FitCSV-Tool.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.
- **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.

Examples

```r
## Not run:
fl <- system.file("extdata/example_files.tar.gz", 
                package = "cycleRtools")
fls <- untar(fl, list = TRUE)
untar(fl) # Extract to working directory.

dat <- lapply(fl, read_ride, format = TRUE, CP = 300, sRPE = 5)

file.remove(fl)
## End(Not run)
```

---

**reset**

Reset a dataset or vector.

Description

if x is a "cycleRdata" object, all columns are reset as appropriate. This can be useful after subsetting a ride dataset, for example. Otherwise, this is a wrapper for x ~ x[[1]].

Usage

```r
reset(x)
```

Arguments

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

Value

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

data(ridedata)

# Remove first minute of data and reset.
data_raw <- ridedata[ridedata$timer.s > 60,]
data_reset <- reset(data_raw)

Description

Formatted cycling data from a Garmin head-unit. Imported via read_fit("file_path", format = TRUE, CP = 310, sRPE = 7). "ridedata" is a typical group ride. "intervaldata" is a session (of sorts) that included two efforts and a cafe stop. The latter is included to demonstrate the use of interval_detect.

Usage

ridedata
intervaldata

Format

An object of class c("cycleRdata", "data.frame"), and additional attributes of CP = 300 & sRPE = 7. The latter are used by several methods in this package. See cycleRdata for a description of columns.

See Also

cycleRdata.

rollmean_

Rolling average smoothing.

Description

Smooth data with a right-aligned (zero-padded) rolling average.

Usage

rollmean_(x, window, ema, narm)

rollmean_smth(data, column, smth.pd, deltat = NULL, ema = FALSE, character.only = FALSE)
rollmean_nunif

Arguments

- **x** numeric; values to be rolled over.
- **window** numeric; size of the rolling window in terms of elements in `x`.
- **ema** logical; should the moving average be exponentially weighted?
- **narm** logical; should NAs be removed?
- **data** a dataset of class `cyclerdata`.
- **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).
- **deltat** the sampling frequency of `data` in seconds per sample; typically 0.5 or 1. If `NULL`, this is estimated.
- **character.only** are column name arguments given as character strings? A backdoor around non-standard evaluation.

Details

`rollmean_nunif` is the core Rcpp function, which rolls over elements in `x` by a window given in `window`; optionally applying exponential weights and/or removing NAs. `rollmean_smth` is a wrapper for `rollmean_nunif` that only has a method for `cyclerdata` objects. The latter will pre-process the data and permits what is effectively the `window` argument being given in time units.

Value

A vector of the same length as the `data[, column]`.

Examples

```r
## Not run:
data(ridedata)

## Smooth power data with a 30 second moving average.
rollmean_smth(ridedata, power,W, 30)

## Or use an exponentially weighted moving average.
rollmean_smth(ridedata, power,W, 30, ema = TRUE)

## End(Not run)
```

---

rollmean_nunif  
*Rolling mean for nonuniform data.*

Description

Produce a rolling average for data sampled at non-uniform time intervals.
smth_plot

Usage

rollmean_nunif(x, t, window)

Arguments

x numeric vector of values to be rolled.
t numeric vector of time values corresponding to elements in x.
window size of the window in terms of t. E.g. 30 (seconds).

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", ..., character.only = 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.
... further arguments to be passed to plot().
character.only are column name arguments given as character strings? A backdoor around non-standard evaluation.

Examples

data(ridedata)

## Plot with a single blue line (default arguments):
smth_plot(ridedata, colour = "blue", main = "Single Colour",
xlab = "Time (seconds)", ylab = "Power (watts)"

## Create some laps.
ridedata$lap <- ceiling(seq(from = 1.1, to = 5, length.out = nrow(ridedata)))
## Plot with lap colours.
smth_plot(ridedata, timer.min, power.W, power.smooth.W, colour = "lap",
xlab = "Time (mins)", ylab = "Power (watts)", main = "Lap Colours")
**summary.cyclerdata**  
*Summary method for cyclerdata class.*

### Description

Relevant summary metrics for cycling data (method for class "cyclerdata").

### Usage

```r
## S3 method for class 'cyclerdata'
summary(object, sRPE = attr(object, "sRPE"),
         CP = attr(object, "CP"), .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; Foster 1998).
- `CP`: optional; Critical Power value (Watts).
- `.smoothpwr`: character string; column name of smoothed power values. Used for xP metric.
- `...`: further arguments passed to or from other methods.

### Value

A list object of class "cyclesummary", which has an associated print method.

### References


### Examples

```r
data(intervaldata)
summary(intervaldata)
```
Summary metrics

Description

Common summary measures of interest to cyclists.

Usage

\begin{verbatim}
ride_time(x, deltat = NULL)
xPower(data)
NP(data)
pwr_TRIMP(data, CP = attr(data, "CP"))
TSS(data, CP = attr(data, "CP"))
\end{verbatim}

Arguments

\begin{itemize}
\item \texttt{x} a vector of time values.
\item \texttt{deltat} numeric; the typical interval between time values, if \texttt{NULL} a best estimate is used.
\item \texttt{data} a "cycleRdata" object, produced from a \texttt{read_ride} function.
\item \texttt{CP} a Critical Power value - e.g. CP or FTP.
\end{itemize}

Details

\texttt{NP} calculates a Normalised Power value. "Normalised Power" is a registered trademark of Peakware Inc.

\texttt{xPower}; Dr. Philip Skiba/Golden Cheetah’s answer to \texttt{NP}.

\texttt{pwr_TRIMP}: Power-Based TRaining IMPulse. Calculates a \textit{normalised} TRIMP value using power data. This is a power-based adaptation of Bannister’s TRIMP, whereby critical power (CP) is assumed to represent 90 to the score associated with one-hour’s riding at CP, to aid interpretation.

\texttt{ride_time} is a simple function for calculating ride time, as opposed to elapsed time.

\texttt{TSS} calculates a Training Stress Score (TSS). TSS is a registered trademark of Peakware Inc.

Value

a single numeric value.

References

Examples

data(ridedata)

## Display all summary metrics with an *apply* call.
fns <- list("ride_time", "xPower", "NP", "pwr_TRIMP", "TSS")
argl <- list(data = ridedata, x = ridedata$timer.s, CP = 300)
metrs <- vapply(fns, function(f) {
  do.call(f, argl[[names(argl) %in% names(formals(f))]])
}, numeric(1))

names(metrs) <- fns
print(metrs)


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). Wbal is a wrapper for the Rcpp function Wbal_.

Usage

Wbal_(t, P, CP)

Wbal(data, time = "timer.s", pwr = "power.smooth.W", CP = attr(data, "CP"), noisy = TRUE, character.only = FALSE)

Arguments

t, P  numeric vectors of time and power, respectively.
CP  a critical power value for use in the calculation.
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.
noisy  logical; create smoother data by pooling power data into sub- and supra-CP sections.
character.only  are column name arguments given as character strings? A backdoor around non-standard evaluation.
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.

It is advisable on physiological grounds to enter smoothed power values to the function, hence this is the default behaviour. If nothing else, this prevents an unrealistic inflation of W’ values that are inconsistent with estimates derived from power-time modelling.

The essence of the algorithm can be seen in the function test file.

Note that if there are NA values in the power column, these are ignored and the corresponding W’ expended value assumes that of the last available power value. NA values are not allowed in the time column.

Value

A numeric vector of W’ balance values, in kilojoules or joules for `wbal` or `wbal_` respectively.

References


See Also

`plot.cycleRdata`.

Examples

```r
## Not run:
data(ridedata)

## Basic usage.
ridedata$Wexp.kJ <- wbal(ridedata, timer.s, power.W, 310)

## Data can be noisy or “smooth”; e.g.
Wbal_noisy <- wbal(ridedata, timer.s, power.W, 310, noisy = TRUE)
Wbal_smth <- wbal(ridedata, timer.s, power.W, 310, noisy = FALSE)

## Plot:
ylim <- rev(extdrange(Wbal_noisy))  # Reverse axes.
plot(ridedata$timer.min, Wbal_noisy, type = "l", ylim = ylim, 
     main = "NOisy")
plot(ridedata$timer.min, Wbal_smth, type = "l", ylim = ylim, 
     main = "Smooth")
```
## 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

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

### Arguments

- **data**: a "cycleRdata" object, produced from a `read_ride` function.  
- **column**: column in `data` giving the values of interest. Needn't be quoted.  
- **binwidth**: how should values in `column` be binned? E.g. `binwidth = 10` will create 10 watt bins if `column` is power data.  
- **zbounds**: optional; a numeric vector of zone boundaries.  
- **character.only**: are column name arguments given as character strings? A backdoor around non-standard evaluation.  
- **...**: arguments to be passed to `barplot()` and/or graphical parameters (`par`).

### Value

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

### Examples

```r
data(ridedata)

## Using power.
zdist_plot(
  data = ridedata, column = power.W,
```
binwidth = 10, # 10 watt bins.
zbounds = c(100, 200, 300),
xlim = c(110, 500), xlab = "Power (Watts)",
main = "Power distribution" # Argument passed to barplot.
)

## Using speed.
zdist_plot(
  data = ridedata, column = speed.kmh,
  binwidth = 2, # 2 km/hr bins.
  zbounds = c(10, 20, 30),
  xlab = "Speed (km/hr)",
  main = "Speed distribution"
)

## Without zone colouring (produces a warning).
zdist_plot(
  data = ridedata, column = speed.kmh,
  binwidth = 5, # 2 km/hr bins.
  xlab = "Speed (km/hr)", main = "Dull"
)

---

**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

```r
data(ridedata)

## Best used to append to existing data.
ridedata$zone <- zone_index(ridedata$power.W, c(100, 200, 300))

## How much distance was covered in each zone?
ridedata$delta.dist <- c(0, diff(ridedata$distance.km))
with(ridedata, tapply(delta.dist, zone, sum, na.rm = TRUE))  # Km.
```

---

### 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, character.only = FALSE)
```

Arguments

- `data`: a "cycleRdata" object, produced from a `read_ride` function.
- `column`: the column name of the data to which the zone boundaries relate.
- `zbounds`: numeric; zone boundaries.
- `pct`: should percentage values be returned?
- `character.only`: 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.

Examples

```
data(ridedata)

## Time spent above and below critical power...
zone_time(ridedata, "power.W", zbounds = 300) / 60  # Minutes.

## Or with more zones...
zone_time(ridedata, "power.W", zbounds = c(100, 200, 300)) / 60

## Or given as a percentage...
zone_time(ridedata, "power.W", zbounds = c(100, 200, 300), pct = TRUE)
```
Index

*Topic datasets
   Pt_prof, 15
   ride_examples, 18

as.cycledata(cycledata), 3
as.Date, 7

ccoef, 14
convert_from_time(convert_time), 2
convert_time, 2
convert_to_time(convert_time), 2
cycledata, 3, 18

diff_section, 4
download_elev_data, 5, 6
elevation_correct, 5, 5

GC, 6
GC_activity(GC), 6
GC_metrics(GC), 6
GC_mmvs(GC), 6

interval_detect, 7, 18
intervaldata(ride_examples), 18
is.cycledata(cycledata), 3

LT, 9

mmv, 7, 10, 11
mmv2, 10, 11

NP(summary_metrics), 22

par, 25
plot.cycledata, 12, 24
predict, 14
predict.Ptmodels, 13, 15
print, 14
Pt_model, 14
Pt_prof, 15

pwr_TRIMP(summary_metrics), 22
read_fit(read_ride), 16
read_pwx(read_ride), 16
read_ride, 3, 7, 16, 22, 25, 27
read_srmm(read_ride), 16
read_tcx(read_ride), 16
reset, 17
ride_examples, 18
ride_time(summary_metrics), 22
ridgedata, 16
ridgedata(ride_examples), 18
rollmean, 18
rollmean_nunif, 19
rollmean_smth(rollmean_), 18

smth_plot, 20
summary, 14
summary.cycledata, 16, 21
summary_metrics, 22
system2, 17
TSS(summary_metrics), 22
Wbal(Wbal_), 23
Wbal_, 23

xPower(summary_metrics), 22
zdist_plot, 25
zone_index, 26
zone_time, 27