Package ‘trackeR’

October 12, 2018

Version 1.2.0

Title Infrastructure for Running, Cycling and Swimming Data from GPS-Enabled Tracking Devices

Description Provides infrastructure for handling running, cycling and swimming data from GPS-enabled tracking devices within R. The package provides methods to extract, clean and organise workout and competition data into session-based and unit-aware data objects of class ‘trackeRdata’ (S3 class). The information can then be visualised, summarised, and analysed through flexible and extensible methods. Frick and Kosmidis (2017) <doi: 10.18637/jss.v082.i07>, which is updated and maintained as one of the vignettes, provides detailed descriptions of the package and its methods, and real-data demonstrations of the package functionality.

Depends R (>= 3.1.0), zoo

Imports ggplot2, ggridges, xml2, RSQLite, jsonlite, raster, scam, foreach, fda, sp, leaflet, ggmap, gridExtra, gtable

Suggests testthat, knitr, rmarkdown, covr

VignetteBuilder knitr

License GPL-3

URL https://github.com/trackerproject/trackeR

BugReports https://github.com/trackerproject/trackeR/issues

RoxygenNote 6.0.1

Encoding UTF-8

LazyData true

NeedsCompilation no

Author Ioannis Kosmidis [aut, cre] (<https://orcid.org/0000-0003-1556-0302>), Hannah Frick [aut] (<https://orcid.org/0000-0002-6049-5258>), Robin Hornak [aut]

Maintainer Ioannis Kosmidis <ioannis.kosmidis@warwick.ac.uk>

Repository CRAN

Date/Publication 2018-10-12 09:00:03 UTC
R topics documented:

append ................................................. 4
append.trackeRdata ..................................... 4
c2d ....................................................... 5
change_units .......................................... 5
change_units.conProfile .................................. 6
change_units.distrProfile .................................. 6
change_units.trackeRdata ................................ 7
change_units.trackeRdataSummary ................................ 7
change_units.trackeRdataZones ................................ 8
change_units.trackeRWprime ................................ 8
collect_units ........................................ 9
compute_breaks ......................................... 9
compute_limits ......................................... 10
concentration_profile ................................... 10
concentration_profile.distrProfile ......................... 11
conversions ........................................... 13
decreasing_smoother ..................................... 17
distance2speed ......................................... 18
distribution_profile ..................................... 19
find_unit_reference_sport .................................. 20
fortify.conProfile ..................................... 21
fortify.distrProfile ................................... 21
fortify.trackeRdata ..................................... 22
fortify.trackeRdataSummary ................................ 22
fortify.trackeRWprime ................................... 23
funPCA .................................................. 23
GC2trackeRdata ......................................... 24
generate_thresholds ..................................... 25
generate_units .......................................... 26
get_operations .......................................... 27
get_operations.conProfile .................................. 27
get_operations.distrProfile .................................. 28
get_operations.trackeRdata ................................ 28
get_profile.distrProfile .................................. 29
get_resting_periods ..................................... 29
get_sport.trackeRWprime ................................ 30
get_units .............................................. 31
get_units.conProfile ................................... 31
get_units.distrProfile ................................... 32
get_units.trackeRdata ................................... 32
get_units.trackeRdataSummary ................................ 33
get_units.trackeRdataZones ................................ 33
generate_thresholds ..................................... 25
get_units.trackeRWprime ................................ 34
impute_speeds ......................................... 34
leaflet_route .......................................... 35
nsessions.trackeRWprime ................................ 36
R topics documented:

plot.conProfile .................................................. 36
plot.distrProfile .................................................. 37
plot.trackeRdata .................................................. 38
plot.trackeRdataSummary ...................................... 39
plot.trackeRdataZones ........................................... 40
plot.trackeRfpca .................................................. 41
plot.trackeRWprime .............................................. 42
plot_route ......................................................... 42
prepare_route ..................................................... 43
prettifyUnit ....................................................... 44
print.trackeRdata ................................................ 45
print.trackeRdataSummary ..................................... 45
profile2fd ........................................................ 46
readX ............................................................. 47
read_container .................................................... 48
read_directory .................................................... 50
ridges ............................................................. 51
ridges.conProfile ................................................. 52
ridges.distrProfile .............................................. 53
ridges.trackeRdata ............................................. 53
run ................................................................. 54
runs ............................................................... 54
sanity_checks ..................................................... 55
scaled ............................................................. 55
scaled.distrProfile .............................................. 56
session_duration .................................................. 56
session_times ..................................................... 57
smoother .......................................................... 57
smoother.conProfile ............................................. 58
smoother.distrProfile .......................................... 58
smoother.trackeRdata ......................................... 59
smoother_control.distrProfile ............................... 60
smoother_control.trackeRdata ............................... 61
sort.trackeRdata .................................................. 61
speed2distance .................................................... 62
summary.trackeRdata .......................................... 62
threshold.trackeRdata .......................................... 64
timeAboveThreshold ............................................ 65
timeline .......................................................... 66
trackeR ........................................................... 67
trackeRdata ...................................................... 67
unique.trackeRdata ............................................. 69
Wexp .............................................................. 70
Wprime .......................................................... 71
zones ............................................................. 73

Index 74
append

Generic function for appending data to existing files

Description

Generic function for appending data to existing files

Usage

append(object, file, ...)

Arguments

object The object to be appended.
file The file to which object is to be appended.
... Arguments to be passed to methods.

append.trackeRdata

Append training sessions to existing file

Description

Append training sessions to existing file

Usage

## S3 method for class 'trackeRdata'
append(object, file, ...)

Arguments

object The object to be appended.
file The file to which object is to be appended.
... Currently not used.
c2d

Transform concentration profile to distribution profile.

**Description**

Transform concentration profile to distribution profile.

**Usage**

```
c2d(cp)
```

**Arguments**

- `cp` Single concentration profile as a zoo object.

---

change_units

Generic function for changing the units of measurement

**Description**

Generic function for changing the units of measurement

**Usage**

```
change_units(object, variable, unit, sport, ...)
```

```
changeUnits(object, variable, unit, sport, ...)
```

**Arguments**

- `object` The object of which the units of measurement are changed.
- `variable` A vector of variables whose units are to be changed.
- `unit` A vector with the units, corresponding to `variable`.
- `sport` A vector of sports (amongst 'cycling', 'running', 'swimming') with each element corresponding to `variable` and `unit`.
- `...` Arguments to be passed to methods.
change_units.conProfile

Change the units of the variables in an conProfile object

Description

Change the units of the variables in an conProfile object

Usage

## S3 method for class 'conProfile'
change_units(object, variable, unit, ...)

Arguments

- object: An object of class conProfile as returned by concentrationProfile.
- variable: A vector of variables to be changed.
- unit: A vector with the units, corresponding to variable.
- ...: Currently not used.

change_units.distrProfile

Change the units of the variables in an distrProfile object

Description

Change the units of the variables in an distrProfile object

Usage

## S3 method for class 'distrProfile'
change_units(object, variable, unit, ...)

Arguments

- object: An object of class distrProfile as returned by distributionProfile.
- variable: A vector of variables to be changed.
- unit: A vector with the units, corresponding to variable.
- ...: Currently not used.
change_units.trackeRdata

Change the units of the variables in an trackeRdata object

Description
Change the units of the variables in an trackeRdata object

Usage
## S3 method for class 'trackeRdata'
change_units(object, variable, unit, sport, ...)

Arguments

- object: An object of class trackeRdata.
- variable: A vector of variables whose units are to be changed.
- unit: A vector with the units, corresponding to variable.
- sport: A vector of sports (amongst 'cycling', 'running', 'swimming') with each element corresponding to variable and unit.
- ...: Arguments to be passed to methods.

change_units.trackeRdataSummary

Change the units of the variables in an trackeRdataSummary object

Description
Change the units of the variables in an trackeRdataSummary object

Usage
## S3 method for class 'trackeRdataSummary'
change_units(object, variable, unit, ...)

Arguments

- object: An object of class trackeRdataSummary.
- variable: A vector of variables to be changed. Note, these are expected to be concepts like 'speed' rather than variable names like 'avgSpeed' or 'avgSpeedMoving'.
- unit: A vector with the units, corresponding to variable.
- ...: Currently not used.
change_units.trackerdataZones

Change the units of the variables in a trackerdataZones object

Description

Change the units of the variables in a trackerdataZones object

Usage

```r
## S3 method for class 'trackerdataZones'
change_units(object, variable, unit, ...)
```

Arguments

- `object`: An object of class `trackerdataZones`.
- `variable`: A vector of variables to be changed. Note, these are expected to be concepts like 'speed' rather than variable names like 'avgSpeed' or 'avgSpeedMoving'.
- `unit`: A vector with the units, corresponding to variable.
- `...`: Currently not used.

change_units.trackerwPrime

Change the units of the variables in a trackerwPrime object

Description

Change the units of the variables in a trackerwPrime object

Usage

```r
## S3 method for class 'trackerwPrime'
change_units(object, variable, unit, ...)
```

Arguments

- `object`: An object of class `trackerwPrime`.
- `variable`: A vector of variables to be changed.
- `unit`: A vector with the units, corresponding to variable.
- `...`: Currently not used.
**collect_units**

*Collect units from the result of* `generate_units`.

**Description**

Collects the units from the results of `generate_units` according to a `unit_reference_sport`.

**Usage**

```r
collect_units(object, unit_reference_sport = NULL)
```

**Arguments**

- `object` a data.frame, as returned by `generate_units`
- `unit_reference_sport` The sport to inherit units from (default is taken to be the most frequent sport in `object`).

---

**compute_breaks**

*Compute a grid of breakpoints per variable from a trackRdata object.*

**Description**

Compute a grid of breakpoints per variable from a `trackRdata` object.

**Usage**

```r
compute_breaks(object, a = 1E-04, n_breaks = 9, limits = NULL, what = c("speed", "heart_rate"))
```

**Arguments**

- `object` A `trackRdata` object.
- `a` The levels at which quantiles will be computed are \( a \) and \( 1 - a \). Default is \( a = 0.0001 \).
- `n_breaks` A scalar determining the number of breakpoints to be computed.
- `limits` A list of a vectors, each specifying the lower and upper limit for each variable to be used when computing the grid. Default is NULL, in which case `compute_limits` is used.
- `what` The variables for which a grid of breakpoints should be computed. Defaults to `c("speed", "heart_rate")`. 
Value
A named list with names as in what, with elements the grids of breakpoints per variable.

Examples
```
data("runs")
compute_breaks(runs, what = c("speed", "heart_rate", "altitude"))
```

compute_limits  

Compute variable limits from a `trackRdata` object.

Description
Compute variable limits from a `trackRdata` object.

Usage
```
compute_limits(object, a = 1e-04)
```

Arguments
- **object**: A `trackRdata` object.
- **a**: The levels at which quantiles will be computed are $a$ and $1 - a$. Default is $a = 0.0001$.

Details
compute_limits computes limits by finding the $a$ and $1 - a$ quantiles for each variable in each session, and then taking the minimum and maximum of the $a$ and $1 - a$, respectively, across sessions.

concentration_profile  

Generic method for concentration profiles

Description
Generic method for concentration profiles

Usage
```
concentration_profile(object, session = NULL, what = NULL, ...)
concentrationProfile(object, session = NULL, what = NULL, ...)
```
Arguments

- **object**: An object of class `trackerdata` or `distrProfile`.
- **session**: A numeric vector of the sessions to be used, defaults to all sessions.
- **what**: The variables for which the distribution profiles should be generated. Defaults to all variables in `object` (what = NULL).
- **...**: Currently not used.

See Also

- `concentration_profile.distrProfile`
- `concentration_profile.trackeRdata`

Examples

```r
## Not run:
## Compute concentration profiles from distribution profiles
data('run', package = 'trackeR')
dProfile <- distributionProfile(run, what = 'speed', grid = seq(0, 12.5, by = 0.05))
cProfile <- concentrationProfile(dProfile)
plot(cProfile, smooth = FALSE)
plot(cProfile)

## And now directly from the 'trackerdata' object, which is a
## considerably faster if all that is needed are the concentration
## profiles
cProfile <- concentrationProfile(runs, what = 'speed',
                               limits = list(speed = c(0, 12.5)))
plot(cProfile, smooth = FALSE)
ridges(cProfile)
plot(cProfile, smooth = TRUE)

## End(Not run)
```

---

**concentration_profile.distrProfile**

*Generate training concentration profiles.*

**Description**

Generate training concentration profiles.

**Usage**

```r
## S3 method for class 'distrProfile'
concentration_profile(object, session = NULL,
                       what = NULL, ...)

## S3 method for class 'trackeRdata'
```
concentration_profile(object, session = NULL,
what = NULL, limits = NULL, parallel = FALSE,
unit_reference_sport = NULL, scale = FALSE, ...)

Arguments

object An object of class trackerdata or distrProfile.
session A numeric vector of the sessions to be used, defaults to all sessions.
what The variables for which the distribution profiles should be generated. Defaults to all variables in object (what = NULL).
... Currently not used.
limits A named list of vectors of two numbers to specify the lowe and upper limits for the variables in what. If NULL (default) the limits for the variables in what are inferred from object.
parallel Logical. Should computation be carried out in parallel? Default is FALSE.
unit_reference_sport The sport to inherit units from (default is taken to be the most frequent sport in object).
scale Logical. If FALSE (default) then the integral of the profiles over the real line matches the session length.

Value

An object of class conProfile.

Object:
A named list with one or more components, corresponding to the value of what. Each component is a matrix of dimension g times n, where g is the length of the grids set in grid (or 200 if grid = NULL) and n is the number of sessions requested in the session argument.

Attributes:
Each conProfile object has the following attributes:

• sport: the sports corresponding to the columns of each list component
• session_times: the session start and end times corresponding to the columns of each list component
• unit_reference_sport: the sport where the units have been inherited from
• operations: a list with the operations that have been applied to the object. See get_operations.distrProfile
• limits: The variable limits that have been used for the computation of the concentration profiles.
• units: an object listing the units used for the calculation of distribution profiles. These is the output of get_units on the corresponding trackerdata object, after inheriting units from unit_reference_sport.
References


---

Auxiliary conversion functions

<table>
<thead>
<tr>
<th>conversions</th>
<th>Auxiliary conversion functions</th>
</tr>
</thead>
</table>

Description

Conversion functions for distance, duration, speed, pace, power, cadence and temperature.

Usage

```r
m2km(variable)
km2m(variable)
m2ft(variable)
ft2m(variable)
m2mi(variable)
mi2m(variable)
km2ft(variable)
ft2km(variable)
km2mi(variable)
mi2km(variable)
ft2mi(variable)
mi2ft(variable)
m2mi(variable)
km2km(variable)
ft2ft(variable)
mi2mi(variable)
```
s2min(variable)
min2s(variable)
s2h(variable)
h2s(variable)
min2h(variable)
h2min(variable)
h2h(variable)
min2min(variable)
s2s(variable)
min2min(variable)
h2h(variable)
degree2degree(variable)

m_per_s2km_per_h(variable)
km_per_h2m_per_s(variable)
m_per_s2ft_per_min(variable)
ft_per_min2m_per_s(variable)
m_per_s2ft_per_s(variable)
ft_per_s2m_per_s(variable)
m_per_s2mi_per_h(variable)
mi_per_h2m_per_s(variable)
m_per_s2km_per_min(variable)
km_per_min2m_per_s(variable)
m_per_s2mi_per_min(variable)
mi_per_min2m_per_s(variable)
conversions

km_per_h2ft_per_min(variable)
ft_per_min2km_per_h(variable)
km_per_h2ft_per_s(variable)
ft_per_s2km_per_h(variable)
km_per_h2mi_per_h(variable)
mi_per_h2km_per_h(variable)
km_per_h2km_per_min(variable)
km_per_min2km_per_h(variable)
km_per_h2mi_per_min(variable)
mi_per_min2km_per_h(variable)
ft_per_min2ft_per_s(variable)
ft_per_s2ft_per_min(variable)
ft_per_min2mi_per_h(variable)
mi_per_h2ft_per_min(variable)
ft_per_min2km_per_min(variable)
km_per_min2ft_per_min(variable)
ft_per_min2mi_per_min(variable)
mi_per_min2ft_per_min(variable)
ft_per_s2mi_per_h(variable)
mi_per_h2ft_per_s(variable)
ft_per_s2km_per_min(variable)
km_per_min2ft_per_s(variable)
ft_per_s2mi_per_min(variable)
mi_per_min2ft_per_s(variable)
mi_per_h2km_per_min(variable)
km_per_min2mi_per_h(variable)
mi_per_h2mi_per_min(variable)
mi_per_min2mi_per_h(variable)
km_per_min2mi_per_min(variable)
mi_per_min2km_per_min(variable)
m_per_s2m_per_s(variable)
km_per_h2km_per_h(variable)
ft_per_min2ft_per_min(variable)
ft_per_s2ft_per_s(variable)
mi_per_h2mi_per_h(variable)
km_per_min2km_per_min(variable)
mi_per_min2mi_per_min(variable)
bpm2bpm(variable)
s_per_m2min_per_km(variable)
min_per_km2s_per_m(variable)
s_per_m2min_per_mi(variable)
min_per_mi2s_per_m(variable)
min_per_km2min_per_mi(variable)
min_per_mi2min_per_km(variable)
min_per_ft2min_per_km(variable)
min_per_ft2min_per_mi(variable)
s_per_m2s_per_m(variable)
min_per_km2min_per_km(variable)
Arguments

variable Variable to be converted.

Description

This smoother ensures a positive response that is a monotone decreasing function of x.
Usage

decreasing_smoother(x, y, k = 30, len = NULL, sp = NULL)
decreasingSmoother(x, y, k = 30, len = NULL, sp = NULL)

Arguments

x The regressor passed on to the formula argument of scam.
y The response passed on to the formula argument of scam.
k Number of knots.
len If NULL, the default, x is used for prediction. Otherwise, prediction is done over the range of x with len equidistant points.
sp A vector of smoothing parameters passed on to scam.

---

distance2speed Convert distance to speed.

Description

Convert distance to speed.

Usage

distance2speed(distance, time, timeunit)

Arguments

distance Distance in meters.
time Time.
timeunit Time unit in speed, e.g., "hours" for speed in *per_h.

Value

Speed in meters per second.
distribution_profile

Generate training distribution profiles.

Description

Generate training distribution profiles.

Usage

distribution_profile(object, session = NULL, what = NULL, grid = NULL, parallel = FALSE, unit_reference_sport = NULL)

distributionProfile(object, session = NULL, what = NULL, grid = NULL, parallel = FALSE, unit_reference_sport = NULL)

Arguments

- **object**: An object of class `trackeRdata`.
- **session**: A numeric vector of the sessions to be used, defaults to all sessions.
- **what**: The variables for which the distribution profiles should be generated. Defaults to all variables in `object` (`what = NULL`).
- **grid**: A named list containing the grid values for the variables in `what`. If `NULL` (default) the grids for the variables in `what` are inferred from `object`.
- **parallel**: Logical. Should computation be carried out in parallel? Default is `FALSE`.
- **unit_reference_sport**: The sport to inherit units from (default is taken to be the most frequent sport in `object`).

Value

An object of class `distrProfile`.

Object:

A named list with one or more components, corresponding to the value of `what`. Each component is a matrix of dimension `g` times `n`, where `g` is the length of the grids set in `grid` (or 201 if `grid = NULL`) and `n` is the number of sessions requested in the `session` argument.

Attributes:

Each `distrProfile` object has the following attributes:

- **sport**: the sports corresponding to the columns of each list component
- **session_times**: the session start and end times corresponding to the columns of each list component
- **unit_reference_sport**: the sport where the units have been inherited from
- **operations**: a list with the operations that have been applied to the object. See `get_operations.distrProfile`
• **limits**: The variable limits that have been used for the computation of the distribution profiles

• **units**: an object listing the units used for the calculation of distribution profiles. These is the output of `get_units` on the corresponding `trackeRdata` object, after inheriting units from `unit_reference_sport`.

**References**


**Examples**

```r
data('run', package = 'trackR')
dProfile <- distribution_profile(run, what = c("speed", "cadence_running"))
## Not run:
plot(dProfile, smooth = FALSE)
## End(Not run)
```

---

**find_unit_reference_sport**

Find the most frequent sport in an object

**Description**

Find the most frequent sport in an object

**Usage**

```r
find_unit_reference_sport(object)
```

**Arguments**

- **object** any object with a `get_sport` method implemented (run methods(get_sport)).
Fortify a `conProfile` object for plotting with ggplot2.

Usage

```r
## S3 method for class 'conProfile'
fortify(model, data, melt = FALSE, ...)
```

Arguments

- `model`: The `conProfile` object.
- `data`: Ignored.
- `melt`: Logical. Should the data be melted into long format instead of the default wide format?
- `...`: Ignored.

Fortify a `distrProfile` object for plotting with ggplot2.

Usage

```r
## S3 method for class 'distrProfile'
fortify(model, data, melt = FALSE, ...)
```

Arguments

- `model`: The `distrProfile` object.
- `data`: Ignored.
- `melt`: Logical. Should the data be melted into long format instead of the default wide format?
- `...`: Ignored.
Fortify a trackeRdata object for plotting with ggplot2

Description
Fortify a trackeRdata object for plotting with ggplot2.

Usage
```r
## S3 method for class 'trackeRdata'
fortify(model, data, melt = FALSE, ...)
```

Arguments
- `model`: The trackeRdata object.
- `data`: Ignored.
- `melt`: Logical. Should the data be melted into long format instead of the default wide format?
- `...`: Ignored.

Fortify a trackeRdataSummary object for plotting with ggplot2.

Description
Fortify a trackeRdataSummary object for plotting with ggplot2.

Usage
```r
## S3 method for class 'trackeRdataSummary'
fortify(model, data, melt = FALSE, ...)
```

Arguments
- `model`: The trackeRdata object.
- `data`: Ignored.
- `melt`: Logical. Should the data be melted into long format instead of the default wide format?
- `...`: Currently not used.
fortify.trackerwprime

Fortify a trackeRWprime object for plotting with ggplot2.

Description

Fortify a trackeRWprime object for plotting with ggplot2.

Usage

```r
# S3 method for class 'trackeRWprime'
fortify(model, data, melt = FALSE, ...)

fortify_trackerwprime(model, data, melt = FALSE, ...)
```

Arguments

- `model` The trackeRWprime object as returned by `Wprime`.
- `data` Ignored.
- `melt` Logical. Should the data be melted into long format instead of the default wide format?
- `...` Ignored.

funPCA

Functional principal components analysis of distribution or concentration profiles.

Description

Functional principal components analysis of distribution or concentration profiles.

Generic function for functional principal components analysis

Usage

```r
# S3 method for class 'distrProfile'
funPCA(object, what, nharm = 4, ...)

# S3 method for class 'conProfile'
funPCA(object, what, nharm = 4, ...)

funPCA(object, ...)
```
Arguments

- object: The object to which a functional principal components analysis is applied.
- what: The variable for which the profiles should be analysed.
- nharm: The number of principal components estimated.
- ...: Arguments to be passed to methods.

Details

The ... argument is passed on to `pca.fd`.

Value

An object of class `trackerfpca`.

References


Examples

```r
## Not run:
data('runs', package = 'trackeR')
dp <- distributionProfile(runs, what = 'speed')
dp.pca <- funPCA(dp, what = 'speed', nharm = 4)
## 1st harmonic captures vast majority of the variation
plot(dp.pca, harm = 1)
## time spent above speed = 0 is the characteristic distinguishing the profiles
sumRuns <- summary(runs)
plot(sumRuns$durationMoving, dp.pca$scores[,1])

## End(Not run)
```

GC2trackeRdata Coercion function for use in Golden Cheetah

Description

Coercion function for use in Golden Cheetah

Usage

```r
GC2trackeRdata(gc, cycling = TRUE, correct_distances = FALSE,
country = NULL, mask = TRUE, from_distances = FALSE, lgap = 30,
lskip = 5, m = 11, silent = FALSE)
```
generate_thresholds

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gc</td>
<td>Output of Gc.activity.</td>
</tr>
<tr>
<td>cycling</td>
<td>Logical. Does the data stem from cycling?</td>
</tr>
<tr>
<td>correct_distances</td>
<td>Logical. Should the distances be corrected for elevation?</td>
</tr>
<tr>
<td>country</td>
<td>ISO3 country code for downloading altitude data. If NULL, country is derived from longitude and latitude</td>
</tr>
<tr>
<td>mask</td>
<td>Logical. Passed on to getData. Should only the altitudes for the specified country be extracted (TRUE) or also those for the neighboring countries (FALSE)?</td>
</tr>
<tr>
<td>from_distances</td>
<td>Logical. Should the speeds be calculated from the distance recordings instead of taken from the speed recordings directly?</td>
</tr>
<tr>
<td>lgap</td>
<td>Time in seconds corresponding to the minimal sampling rate.</td>
</tr>
<tr>
<td>lskip</td>
<td>Time in seconds between the last observation before a small break and the first imputed speed or the last imputed speed and the first observation after a small break.</td>
</tr>
<tr>
<td>m</td>
<td>Number of imputed observations in each small break.</td>
</tr>
<tr>
<td>silent</td>
<td>Logical. Should warnings be generated if any of the sanity checks on the data are triggered?</td>
</tr>
</tbody>
</table>

See Also

trackerdata

generate_thresholds Generate default thresholds.

Description

Generate default thresholds.

Usage

generate_thresholds(variable, lower, upper, sport, ...)

generateDefaultThresholds(variable, lower, upper, sport, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td>A vector of variables with user-specified thresholds.</td>
</tr>
<tr>
<td>lower</td>
<td>A vector of lower limits corresponding to the elements of variable.</td>
</tr>
<tr>
<td>upper</td>
<td>A vector of upper limits corresponding to the elements of variable.</td>
</tr>
<tr>
<td>sport</td>
<td>A vector of sports (amongst 'cycling', 'running', 'swimming') with each element corresponding to variable, lower and upper.</td>
</tr>
<tr>
<td>...</td>
<td>Currently not used.</td>
</tr>
</tbody>
</table>
generate_units

Generate and set base units.

Description

Generate and set base units.

Usage

generate_units(variable, unit, sport, ...)

generateBaseUnits(variable, unit, sport, ...)

Arguments

variable A vector of variables with user-specified units.
unit A vector with the user-specified units, corresponding to variable (see details).
sport A vector of sports (amongst 'cycling', 'running', 'swimming') with each element corresponding to variable and unit.
... Currently not used.

Details

The available units are

- variables latitude and longitude with unit degree (default)
- variables altitude, distance with unit m (default), km, mi or ft
- variable heart_rate with unit bpm (default)
- variable speed with unit m_per_s (default), km_per_h, ft_per_min, ft_per_s or mi_per_h
- variable cadence_running with unit steps_per_min (default; running only)
- variable cadence_cycling with unit rev_per_min (default; cycling only)
- variable power with unit W (Watt; default) or kW (cycling only)
- variable temperature with unit C (Celsius; default) or F

Note that generate_units checks if the supplied combinations of variable and sport are valid. generate_units will not check if any of the supplied units are correct for the corresponding combination of variable and sport.
**get_operations**

Generic function for retrieving the operation settings

**Description**

Generic function for retrieving the operation settings

**Usage**

```r
get_operations(object, ...)  
getOperations(object, ...)
```

**Arguments**

- **object**
  - The object of which the units of measurement are retrieved.
- **...**
  - Arguments to be passed to methods.

---

**get_operations.conProfile**

Get the operation settings of an conProfile object

**Description**

Get the operation settings of an conProfile object

**Usage**

```r
## S3 method for class 'conProfile'
get_operations(object, ...)
```

**Arguments**

- **object**
  - An object of class conProfile as returned by concentrationProfile.
- **...**
  - Currently not used.
get_operations.distrProfile

Get the operation settings of an distrProfile object

Description

Get the operation settings of an distrProfile object

Usage

## S3 method for class 'distrProfile'
get_operations(object, ...)

Arguments

object An object of class distrProfile as returned by distributionProfile.
... Currently not used.

get_operations.trackeRdata

Get the operation settings of an trackeRdata object

Description

Get the operation settings of an trackeRdata object

Usage

## S3 method for class 'trackeRdata'
get_operations(object, ...)

Arguments

object An object of class trackeRdata.
... Currently not used.
get_profile.distrProfile

Generic function to subset distribution and concentration profiles

Description

Generic function to subset distribution and concentration profiles

Usage

## S3 method for class 'distrProfile'
get_profile(object, session = NULL, what = NULL, ...)

## S3 method for class 'conProfile'
get_profile(object, session = NULL, what = NULL, ...)

get_profile(object, session, what, ...)

Arguments

- **object**: An object of class `distrProfile` or `conProfile` as returned by `distribution_profile` and `concentration_profile`, respectively.
- **session**: A numeric vector of the sessions to selected. Defaults to all sessions.
- **what**: A character version of the variables to be selected. Defaults to all variables in `object` (what = NULL).
- **...**: Current no used.

get_resting_periods

Extract resting period characteristics

Description

Extract resting period characteristics

Usage

get_resting_periods(times, session_threshold)

restingPeriods(times, session_threshold)

Arguments

- **times**: Timestamps.
- **session_threshold**: The threshold in hours for the time difference between consecutive timestamps above which they are considered to belong to different training sessions.
Value

A list containing a dataframe with start, end, and duration for each session and the resting time between sessions, named `sessions` and `restingTime`, respectively.

---

`get_sport.trackerWprime`

Generic function for extracting sports

Description

Generic function for extracting sports

Usage

```r
# S3 method for class 'trackerWprime'
get_sport(object, ...)  

# S3 method for class 'conProfile'
get_sport(object, session = NULL, ...)  

get_sport(object, session = NULL, ...)  

get_sport(object, session = NULL, ...)  

get_sport(object, session = NULL, ...)  

get_sport(object, session = NULL, ...)  
```

Arguments

- `object` The object from which to extract sports.
- `...` Arguments to be passed to methods.
- `session` The sessions for which to extract sports.
get_units

---

**get_units**  
*Generic function for extracting the units of measurement*

---

**Description**

Generic function for extracting the units of measurement

**Usage**

```r
get_units(object, ...)  
getUnits(object, ...)
```

**Arguments**

- `object`: The object of which the units of measurement are retrieved.
- `...`: Arguments to be passed to methods.

---

**get_units.conProfile**  
*Get the units of the variables in an conProfile object*

---

**Description**

Get the units of the variables in an conProfile object

**Usage**

```r
## S3 method for class 'conProfile'
get_units(object, ...)
```

**Arguments**

- `object`: An object of class conProfile.
- `...`: Currently not used.
Get the units of the variables in an `distrProfile` object

**Description**

Get the units of the variables in an `distrProfile` object

**Usage**

```r
## S3 method for class 'distrProfile'
get_units(object, ...)
```

**Arguments**

- `object` An object of class `distrProfile`.
- `...` Currently not used.

---

Get the units of the variables in an `trackeRdata` object

**Description**

Get the units of the variables in an `trackeRdata` object

**Usage**

```r
## S3 method for class 'trackeRdata'
get_units(object, ...)
```

**Arguments**

- `object` An object of class `trackeRdata`.
- `...` Currently not used.
get_units.trackerdataSummary

*Get the units of the variables in an trackerdataSummary object*

---

**Description**

Get the units of the variables in an trackerdataSummary object

**Usage**

```r
## S3 method for class 'trackerdataSummary'
get_units(object, ...)
```

**Arguments**

- `object`: An object of class trackerdataSummary.
- `...`: Currently not used.

---

get_units.trackerdataZones

*Get the units of the variables in an trackerdataZones object*

---

**Description**

Get the units of the variables in an trackerdataZones object

**Usage**

```r
## S3 method for class 'trackerdataZones'
get_units(object, ...)
```

**Arguments**

- `object`: An object of class trackerdataZones.
- `...`: Currently not used.
get_units.trackerwprime

Get the units of the variables in a trackerwprime object

Description
Get the units of the variables in a trackerwprime object

Usage
### S3 method for class 'trackerwprime'
get_units(object, ...)

Arguments
- object: An object of class trackerwprime.
- ...: Currently not used.

impute_speeds

Impute speeds

Description
Impute speeds of 0 during small breaks within a session.

Usage
impute_speeds(session_data, from_distances = TRUE, lgap = 30, lskip = 5, m = 11, sport = "cycling", units = NULL)
imputeSpeeds(session_data, from_distances = TRUE, lgap = 30, lskip = 5, m = 11, sport = "cycling", units = NULL)

Arguments
- session_data: A multivariate zoo object with observations of either distance or speed (named Distance or Speed, respectively).
- from_distances: Logical. Should the speeds be calculated from the distance recordings instead of taken from the speed recordings directly?
- lgap: Time in seconds corresponding to the minimal sampling rate.
- lskip: Time in seconds between the last observation before a small break and the first imputed speed or the last imputed speed and the first observation after a small break.
- m: Number of imputed observations in each small break.
sport

What sport does sessions_data contain data of? Either 'cycling' (default), 'running', 'swimming'.

units

Units of measurement.

Value

A multivariate zoo object with imputed observations: 0 for speed, last known position for latitude, longitude and altitude, NA for all other variables. Distances are calculated based on speeds after imputation.

References


**leaflet_route**

Plot routes for training sessions

Description

Plot the route ran/cycled during training on an interactive map. Internet connection is required to download the background map. Icons are by Maps Icons Collection [https://mapicons.mapsmarker.com](https://mapicons.mapsmarker.com)

Usage

```r
leaflet_route(x, session = NULL, threshold = TRUE, ...) 
```

```r
leafletRoute(x, session = NULL, threshold = TRUE, ...) 
```

Arguments

- `x` A object of class `trackeRdata`.
- `session` A numeric vector of the sessions to be plotted. Defaults to all sessions.
- `threshold` Logical. Should thresholds be applied?
- `...` Additional arguments passed on to `threshold`.

Examples

```r
## Not run:
data('runs', package = 'trackeR')
leafletRoute(runs, session = 23:24)
```

## End(Not run)
Generic function for calculating number of sessions

**Description**

Generic function for calculating number of sessions

**Usage**

```r
## S3 method for class 'trackerWprime'
nsessions(object, ...)

## S3 method for class 'distrProfile'
nsessions(object, ...)

## S3 method for class 'conProfile'
nsessions(object, ...)

## S3 method for class 'trackerDdataSummary'
nsessions(object, ...)
```

**Arguments**

- `object`: The object for which to calculate the number of sessions.
- `...`: Arguments to be passed to methods.

---

**plot.conProfile**

Plot concentration profiles.

**Description**

Plot concentration profiles.

**Usage**

```r
## S3 method for class 'conProfile'
plot(x, session = NULL, what = NULL,
     multiple = FALSE, smooth = FALSE, ...)
```
plot.distrProfile

Arguments

- **x**: An object of class `conProfile` as returned by `concentration_profile`.
- **session**: A numeric vector of the sessions to be plotted, defaults to all sessions.
- **what**: Which variables should be plotted? Defaults to all variables in object (what = NULL).
- **multiple**: Logical. Should all sessions be plotted in one panel?
- **smooth**: Logical. Should unsmoothed profiles be smoothed before plotting?
- ... Further arguments to be passed to `smoother_control.distrProfile`.

Examples

data('runs', package = 'trackeR')
dProfile <- distributionProfile(runs, session = 1:3, what = 'speed',
   grid = seq(0, 12.5, by = 0.05))
cProfile <- concentrationProfile(dProfile)
## Not run:
plot(cProfile, smooth = FALSE)
plot(cProfile)
## End(Not run)

---

plot.distrProfile  
Plot distribution profiles.

Description

Plot distribution profiles.

Usage

## S3 method for class 'distrProfile'
plot(x, session = NULL, what = NULL,
   multiple = FALSE, smooth = FALSE, ...)

Arguments

- **x**: An object of class `distrProfile` as returned by `distribution_profile`.
- **session**: A numeric vector of the sessions to be plotted, defaults to all sessions.
- **what**: Which variables should be plotted? Defaults to all variables in object (what = NULL).
- **multiple**: Logical. Should all sessions be plotted in one panel?
- **smooth**: Logical. Should unsmoothed profiles be smoothed before plotting?
- ... Further arguments to be passed to `smoother_control.distrProfile`.
Examples

```r
## Not run:
data('runs', package = 'trackeR')
dProfile <- distribution_profile(runs, session = 1:2,
    what = "speed", grid = seq(0, 12.5, by = 0.05))
plot(dProfile, smooth = FALSE)
plot(dProfile, smooth = FALSE, multiple = TRUE)
plot(dProfile, multiple = TRUE)

## End(Not run)
```

plot.trackerdata  

### Plot training sessions in form of trackerdata objects

Description

Plot training sessions in form of trackerdata objects

Usage

```r
## S3 method for class 'trackerdata'
plot(x, session = NULL, what = c("pace", "heart_rate"),
threshold = TRUE, smooth = FALSE, trend = TRUE,
dates = TRUE, unit_reference_sport = NULL, moving_threshold = NULL, ...)
```

Arguments

- **x**: An object of class trackerdata.
- **session**: A numeric vector of the sessions to be plotted, defaults to all sessions.
- **what**: Which variables should be plotted?
- **threshold**: Logical. Should thresholds be applied?
- **smooth**: Logical. Should the data be smoothed?
- **trend**: Logical. Should a smooth trend be plotted?
- **dates**: Logical. Should the date of the session be used in the panel header?
- **unit_reference_sport**: The sport to inherit units from (default is taken to be the most frequent sport in object).
- **moving_threshold**: A named vector of 3 speeds to be used for thresholding pace, given in the unit of the speed measurements in object. If NULL (default), the speeds are taken to be c(cycling = 2, running = 1, swimming = 0.5). See Details.
- **...**: Further arguments to be passed to threshold and smootherControl.trackerdata.
Details

Note that a threshold is always applied to the pace. This (upper) threshold corresponds to a speed of 1.4 meters per second, the preferred walking speed of humans. The lower threshold is 0.

The units for the variables match those of the sport specified by unit_reference_sport.

Examples

```r
## Not run:
data('runs', package = 'trackeR')
## plot heart rate and pace for the first 3 sessions
plot(runs, session = 1:3)
## plot raw speed data for session 4
plot(runs, session = 4, what = "speed", threshold = FALSE, smooth = FALSE)
## threshold speed variable
plot(runs, session = 4, what = "speed", threshold = TRUE, smooth = FALSE,
     variable = "speed", lower = 0, upper = 10)
## and smooth (thresholding with default values)
plot(runs, session = 4, what = "speed", threshold = TRUE,
     smooth = TRUE, width = 15, parallel = FALSE)
```

plot.trackeRdataSummary

Plot an object of class `trackeRdataSummary`.

Description

Plot an object of class `trackeRdataSummary`.

Usage

```r
## S3 method for class 'trackeRdataSummary'
plot(x, date = TRUE, what = NULL,
     group = NULL, trend = TRUE, 
     ...)```

Arguments

- `x`: An object of class `trackeRdataSummary`.
- `date`: Should the date or the session number be used on the abscissa?
- `what`: Name of variables which should be plotted. Default is all.
- `group`: Which group of variables should be plotted? This can either be total or moving. Default is both.
- `trend`: Should a smooth trend be plotted?
- `...`: Currently not used.
plot.trackeRdataZones

Description

Plot training zones.

Usage

## S3 method for class 'trackeRdataZones'
plot(x, percent = TRUE, ...)

Arguments

x An object of class trackeRdataZones as returned by zones.

percent Logical. Should the relative or absolute times spent training in the different zones be plotted?

... Currently not used.

Examples

## Not run:
data('runs', package = 'trackeR')
runSummary <- summary(runs)
plot(runSummary)
plot(runSummary, date = FALSE, group = 'total',
     what = c('distance', 'duration', 'avgSpeed'))

## End(Not run)
plot.trackeRfpca

Plot function for functional principal components analysis of distribution and concentration profiles.

Description

Plot function for functional principal components analysis of distribution and concentration profiles.

Usage

```r
## S3 method for class 'trackeRfpca'
plot(x, harm = NULL, expand = NULL, 
     pointplot = TRUE, ...)
```

Arguments

- `x`: An object of class `trackeRfpca` as returned by `funPCA`.
- `harm`: A numerical vector of the harmonics to be plotted. Defaults to all harmonics.
- `expand`: The factor used to generate suitable multiples of the harmonics. If `NULL`, the effect of +/- 2 standard deviations of each harmonic is plotted.
- `pointplot`: Should the harmonics be plotted with + and - point characters? Otherwise, lines are used.
- `...`: Currently not used.

References


See Also

- `plot.pca.fd`

Examples

```r
## Not run:
data('runs', package = 'trackeR')
dp <- distributionProfile(runs, what = 'speed')
dp.pca <- funPCA(dp, what = 'speed', nharm = 4)
## 1st harmonic captures vast majority of the variation
plot(dp.pca)
plot(dp.pca, harm = 1, pointplot = FALSE)
## End(Not run)
```
**plot_trackRwprime**

*Plot W'*.

### Description

Plot W'.

### Usage

```r
## S3 method for class 'trackRwprime'
plot(x, session = NULL, dates = TRUE,
     scaled = TRUE, ...)
```

### Arguments

- **x**: An object of class trackRwprime as returned by `Wprime`.
- **session**: A numeric vector of the sessions to be plotted, defaults to all sessions.
- **dates**: Logical. Should the date of the session be used in the panel header?
- **scaled**: Logical. Should the W' be scaled to the movement variable (power or speed) which is then plotted in the background?
- **...**: Currently not used.

### Examples

```r
## Not run:
data('runs', package = 'trackR')
wexp <- Wprime(runs, session = 1:3, cp = 4, version = '2012')
plot(wexp, session = 1:2)
## End(Not run)
```

---

**plot_route**

*Plot routes for training sessions*

### Description

Plot the route ran/cycled during training onto a background map. Internet connection is required to download the background map.

### Usage

```r
plot_route(x, session = 1, zoom = NULL, speed = TRUE, threshold = TRUE,
           mfrow = NULL, ...)

plotRoute(x, session = 1, zoom = NULL, speed = TRUE, threshold = TRUE,
           mfrow = NULL, ...)
```
Arguments

x A object of class `trackeRdata`.

session A numeric vector of the sessions to be plotted. Defaults to the first session, all sessions can be plotted by `session = NULL`.

zoom The zoom level for the background map as passed on to `get_map` (2 corresponds roughly to continent level and 20 to building level).

speed Logical. Should the trace be colored according to speed?

threshold Logical. Should thresholds be applied?

mfrow A vector of 2 elements, number of rows and number of columns, specifying the layout for multiple sessions.

... Additional arguments passed on to `threshold` and `get_map`, e.g., `source` and `maptype`.

See Also

`get_map`, `ggmap`

Examples

```r
## Not run:
data('runs', package = 'trackeR')
plotRoute(runs, session = 4, zoom = 13)
plotRoute(runs, session = 4, zoom = 13, maptype = "hybrid")
## multiple sessions
plotRoute(runs, session = c(1:5, 8:11), source = "google")
## different zoom level per panel
plotRoute(runs, session = 6:7, source = "google", zoom = c(13, 14))
```

---

**prepare_route**

Prepare a data.frame for use in leaflet_route and plot_route

Description

Prepare a data.frame for use in leaflet_route and plot_route

Usage

```r
prepare_route(x, session = 1, threshold = TRUE, ...)
```

Arguments

x a trackeRdata object.

session which session to prepare the data.frame for?

threshold if TRUE (default), then thresholds are applied to x prior to preparing the data.frame.

... Additional arguments to be passed to threshold.
Details

To be used internally in mapping function and rarely by the user.

Value

A data.frame with variables longitude, latitude, speed, SessionID, longitude0, longitude1, latitude0, latitude1. The observations are ordered according to the timestamp they have in x. A suffix of 0 indicates 'start' and a suffix of 1 indicates 'end' at any given observation.

---

prettifyUnit  

*Returns 'pretty' units for use for plotting or printing*

---

Description

Returns 'pretty' units for use for plotting or printing

Usage

```r
prettifyUnit(unit)
prettifyUnits(unit)
```

Arguments

- `unit`: a unit as recorded in the data.frame generated by `generate_units`.

Details

`prettifyUnits` is the vectorized version of `prettifyUnit`.

Examples

```r
prettifyUnit("m_per_s")
prettifyUnit("rev_per_min")
prettifyUnits(c("rev_per_min", "ft_per_min"))
```
Description

print method for \texttt{trackRdata} objects

Usage

\texttt{## S3 method for class 'trackRdata'}
\texttt{print(x, duration\_unit = "h", digits = 2, \ldots )}

Arguments

- \texttt{x}: An object of class \texttt{trackRdata}.
- \texttt{duration\_unit}: The unit of duration in the resulting output. Default is \texttt{h} (hours).
- \texttt{digits}: Number of digits to be printed.
- \ldots: Currently not used; only for compatibility with generic \texttt{summary} method only.

Details

The print method returns training coverage, number of sessions and total training duration from the data in the \texttt{trackRdata} object.

Description

Print method for session summaries.

Usage

\texttt{## S3 method for class 'trackRdataSummary'}
\texttt{print(x, \ldots, digits = 2)}

Arguments

- \texttt{x}: An object of class \texttt{trackRdataSummary}.
- \ldots: Not used, for compatibility with generic summary method only.
- \texttt{digits}: Number of digits to be printed.
profile2fd

Transform distribution and concentration profiles to functional data objects of class fd.

Description

Transform distribution and concentration profiles to functional data objects of class fd.

Usage

profile2fd(object, what, ...)

Arguments

- **object**: An object of class distrProfile or conProfile, as returned by distributionProfile and concentrationProfile, respectively.
- **what**: The variable for which the profiles should be transformed to a functional data object.
- **...**: Additional arguments passed on to Data2fd

Value

An object of class fd.

Examples

```r
## Not run:
library('fda')
data('runs', package = 'trackR')
dp <- distributionProfile(runs, what = 'speed')
dpFun <- profile2fd(dp, what = 'speed',
    fdnames = list('speed', 'sessions', 'time above threshold'))
dp.pca <- pca(dpFun, nharm = 4)
## 1st harmonic captures vast majority of the variation
dp.pca$varprop
## time spent above speed = 0 is the characteristic distinguishing the profiles
plot(dp.pca, harm = 1)
sumRuns <- summary(runs)
plot(sumRuns$durationMoving, dp.pca$scores[,1])

## End(Not run)
```
readX

Read a training file in tcx, gpx, db3 or Golden Cheetah’s JSON format

Description

Read a training file in tcx, gpx, db3 or Golden Cheetah’s JSON format

Usage

readTCX(file, timezone = "", speedunit = "m_per_s", distanceunit = "m", ...)
readGPX(file, timezone = "", speedunit = "km_per_h", distanceunit = "km", ...)
readDB3(file, timezone = "", table = "gps_data", speedunit = "km_per_h", distanceunit = "km", ...)
readJSON(file, timezone = "", speedunit = "km_per_h", distanceunit = "km", ...)

Arguments

file The path to the file.
timezone The timezone of the observations as passed on to as.POSIXct. Ignored for JSON files.
speedunit Character string indicating the measurement unit of the speeds in the container file to be converted into meters per second. See Details.
distanceunit Character string indicating the measurement unit of the distance in the container file to be converted into meters. See Details.
... Currently not used.
table Character string indicating the name of the table with the GPS data in the db3 container file.

Details

Available options for speedunit currently are km_per_h, m_per_s, mi_per_h, ft_per_min and ft_per_s. The default is m_per_s for TCX files and km_per_h for db3 and Golden Cheetah’s json files. Available options for distanceunit currently are km, m, mi and ft. The default is m for TCX and km for gpx, db3 and Golden Cheetah’s json files.

readTCX, readGPX, readGPX and readDB3, try to identify the sport from the data in the container file. If that fails, then an attempt is made to guess the sport from keywords in the filename. If identification is not possible then the file attribute of the returned object has value NA.

Reading Golden Cheetah’s JSON files is experimental.
Examples

```r
## read raw data
filepath <- system.file("extdata/tcx", "2013-06-08-090442.TCX", package = "trackerR")
run0 <- readTcx(filepath = filepath, timezone = "GMT")

## turn into trackerRdata object
units0 <- generate_units()
run0 <- trackerRdata(run0, units = units0)

## alternatively
## Not run:
run0 <- read_container(filepath, type = "tcx", timezone = "GMT")

## End(Not run)
```

read_container

Read a GPS container file.

Description

Read a GPS container file.

Usage

```r
read_container(file, type = c("tcx", "gpx", "db3", "json"),
    table = "gps_data", timezone = "", session_threshold = 2,
    correct_distances = FALSE, country = NULL, mask = TRUE,
    from_distances = NULL, speedunit = NULL, distanceunit = NULL,
    sport = NULL, lgap = 30, lskip = 5, m = 11, silent = FALSE)
```

Arguments

- **file**: The path to the file.
- **type**: The type of the GPS container file. Supported so far are tcx, db3, and json.
- **table**: The name of the table in the database if type is set to db3, ignored otherwise.
- **timezone**: The timezone of the observations as passed on to `as.POSIXct`. Ignored for JSON files.
- **session_threshold**: The threshold in hours for the time difference between consecutive timestamps above which they are considered to belong to different training sessions.
**correct_distances**
Logical. Should the distances be corrected for elevation?

**country**
ISO3 country code for downloading altitude data. If NULL, country is derived from longitude and latitude.

**mask**
Logical. Passed on to `getData`. Should only the altitudes for the specified country be extracted (TRUE) or also those for the neighboring countries (FALSE)?

**from_distances**
Logical. Should the speeds be calculated from the distance recordings instead of taken from the speed recordings directly. Defaults to TRUE for tcx and Golden Cheetah’s json files and to FALSE for db3 files.

**speedunit**
Character string indicating the measurement unit of the speeds in the container file to be converted into meters per second. Default is m_per_s when type is tcx and km_per_h when type is db3 or json. See Details.

**distanceunit**
Character string indicating the measurement unit of the distance in the container file to be converted into meters. Default is m when type is tcx and km when type is db3 or json. See Details.

**sport**
What sport does file contain data from? Either ‘cycling’, ‘running’, ‘swimming’ or NULL (default), in which case the sport is directly obtained from the readX extractors.

**lgap**
Time in seconds corresponding to the minimal sampling rate.

**lskip**
Time in seconds between the last observation before a small break and the first imputed speed or the last imputed speed and the first observation after a small break.

**m**
Number of imputed observations in each small break.

**silent**
Logical. Should warnings be generated if any of the sanity checks on the data are triggered?

### Details
Available options for `speedunit` currently are km_per_h, m_per_s, mi_per_h, ft_per_min and ft_per_s. Available options for `distanceunit` currently are km, m, mi and ft.

`read_container` try to identify the sport from the data in the container file. If that fails, then an attempt is made to guess the sport from keywords in the filename. If identification is not possible then an error is returned from `trackeRdata`. To avoid that error, and if the sport is known, append an appropriate keyword to the filename (e.g. ‘ride’, ‘swim’, ‘run’). This should fix the error.

### Value
An object of class `trackeRdata`.

### See Also
`trackeRdata`, `readTCX`, `readDB3`, `readJSON`

### Examples
```r
correct_distances = TRUE
country = 'US'
mask = TRUE
from_distances = FALSE
speedunit = 'm_per_s'
distanceunit = 'km'
sport = NA
lgap = 0
lskip = 0
m = 0
silent = FALSE

filepath <- system.file("extdata/tcx", "2013-06-08-090442.TCX", package = "trackeR")
run <- read_container(filepath, type = "tcx", timezone = "GMT")
```
Read all supported container files from a supplied directory

Description

Read all supported container files from a supplied directory

Usage

read_directory(directory, aggregate = FALSE, table = "gps_data", timezone = ",", session_threshold = 2, correct_distances = FALSE, country = NULL, mask = TRUE, from_distances = NULL, speedunit = list(tcx = "m_per_s", gpx = "km_per_h", db3 = "km_per_h", json = "km_per_h"), distanceunit = list(tcx = "m", gpx = "km", db3 = "km", json = "km"), sport = NULL, lgap = 30, lskip = 5, m = 11, silent = FALSE, parallel = FALSE, verbose = TRUE)

read_directory(directory, aggregate = FALSE, table = "gps_data", timezone = ",", session_threshold = 2, correct_distances = FALSE, country = NULL, mask = TRUE, from_distances = NULL, speedunit = list(tcx = "m_per_s", gpx = "km_per_h", db3 = "km_per_h", json = "km_per_h"), distanceunit = list(tcx = "m", gpx = "km", db3 = "km", json = "km"), sport = NULL, lgap = 30, lskip = 5, m = 11, silent = FALSE, parallel = FALSE, verbose = TRUE)

Arguments

directory The path to the directory.
agegregate Logical. Aggregate data from different files to the same session if observations are less then session_threshold hours apart? Alternatively, data from different files is stored in different sessions.
table The name of the table in the database for db3 files.
timezone The timezone of the observations as passed on to as.POSIXct. Ignored for JSON files.
session_threshold The threshold in hours for the time difference between consecutive timestamps above which they are considered to belong to different training sessions.
correct_distances Logical. Should the distances be corrected for elevation?
country ISO3 country code for downloading altitude data. If NULL, country is derived from longitude and latitude
mask Logical. Passed on to getData. Should only the altitudes for the specified country be extracted (TRUE) or also those for the neighboring countries (FALSE)?
from_distances Logical. Should the speeds be calculated from the distance recordings instead of taken from the speed recordings directly. Defaults to TRUE for tcx and Golden Cheetah’s json files and to FALSE for db3 files.
speedunit Character string indicating the measurement unit of the speeds in the container file to be converted into meters per second. Default is m_per_s for tcx files and km_per_h for db3 and Golden Cheetah’s json files. See Details.

distanceunit Character string indicating the measurement unit of the distance in the container file to be converted into meters. Default is m for tcx files and km for db3 and Golden Cheetah’s json files. See Details.
sport What sport do the files in directory correspond to? Either 'cycling', 'running', 'swimming' or NULL (default), in which case an attempt is made to extract the sport from each file in directory.
lgap Time in seconds corresponding to the minimal sampling rate.
1skip Time in seconds between the last observation before a small break and the first imputed speed or the last imputed speed and the first observation after a small break.
m Number of imputed observations in each small break.
silent Logical. Should warnings be generated if any of the sanity checks on the data are triggered?
parallel Logical. Should reading be carried out in parallel? If TRUE reading is performed in parallel using the backend provided to foreach. Default is FALSE.
verbose Logical. Should progress reports be printed?

Details

Available options for speedunit currently are km_per_h, m_per_s, mi_per_h, ft_per_min and ft_per_s. Available options for distanceunit currently are km, m, mi and ft.

If aggregate = TRUE, then if sport = NULL the sport in all sessions is determined by the first file read with a sport specification; else if sport is one of the other valid options it determines the sport for all sessions.

Value

An object of class trackerdata.

See Also

trackerdata, readTCX, readDB3, readJSON

ridges

Generic function for ridgeline plots

Description

Generic function for ridgeline plots
Usage

ridges(x, ...)

Arguments

x An object of class distrProfile or conProfile.

... Arguments to be passed to methods.

See Also

ridges.trackeRdata ridges.conProfile ridges.distrProfile

---

ridges.conProfile  Ridgeline plots for distrProfile objects

Description

Ridgeline plots for distrProfile objects

Usage

## S3 method for class 'conProfile'
ridges(x, session = NULL, what = NULL,
       smooth = FALSE, ...)

Arguments

x An object of class conProfile as returned by concentration_profile.

session A numeric vector of the sessions to be plotted, defaults to all sessions.

what Which variables should be plotted? Defaults to all variables in object (what = NULL).

smooth Logical. Should unsmoothed profiles be smoothed before plotting?

... Further arguments to be passed to smoother_control.distrProfile.

Examples

## Not run:

data('runs', package = 'trackeR')
dProfile <- distributionProfile(runs, what = c('speed', 'heart_rate'))
cProfile <- concentrationProfile(dProfile)
ridges(cProfile, what = "speed")
ridges(cProfile, what = "heart_rate")

## End(Not run)
ridges.distrProfile  
Ridgeline plots for distrProfile objects

Description
Ridgeline plots for distrProfile objects

Usage
## S3 method for class 'distrProfile'
ridges(x, session = NULL, what = NULL,
       smooth = FALSE, ...)

Arguments
x  An object of class distrProfile as returned by distribution_profile.
session A numeric vector of the sessions to be plotted, defaults to all sessions.
what Which variables should be plotted? Defaults to all variables in object (what = NULL).
smooth Logical. Should unsmoothed profiles be smoothed before plotting?
... Further arguments to be passed to smoother_control.distrProfile.

Examples
## Not run:

data('runs', package = 'trackeR')
dProfile <- distribution_profile(runs, what = c("speed", "heart_rate"))
ridges(dProfile)

## End(Not run)

ridges.trackeRdata  
Ridgeline plots for trackeRdata objects

Description
Ridgeline plots for trackeRdata objects

Usage
## S3 method for class 'trackeRdata'
ridges(x, session = NULL, what = "speed",
       smooth = TRUE, ...)
Arguments

x  A `trackeRdata` object.
session  A numeric vector of the sessions to be used, defaults to all sessions.
what  The variables for which the distribution profiles should be generated. Defaults to all variables in object (what = NULL).
smooth  Logical. Should the concentration profiles be smoothed before plotting?
...  Currently not used.

Examples

```r
## Not run:
data('runs', package = 'trackeR')
ridges(runs)

## End(Not run)
```

Description

Training session.

Usage

run

Format

A `trackeRdata` object containing one running training session.

Description

Training sessions.

Usage

runs

Format

A `trackeRdata` object containing 33 running training sessions.
### sanity_checks

**Sanity checks for tracking data**

#### Description

Heart rate measurements of 0 are set to NA, assuming the athlete is alive. Observations with missing or duplicated time stamps are removed.

#### Usage

```r
sanity_checks(dat, silent)
```

#### Arguments

- `dat` : Data set to be cleaned up.
- `silent` : Logical. Should warnings be generated if any of the sanity checks on the data are triggered?

### scaled

**Generic function for scaling**

#### Description

Generic function for scaling

#### Usage

```r
scaled(object, ...)
```

#### Arguments

- `object` : The object to be scaled.
- `...` : Arguments to be passed to methods.
scaled.distrProfile  
Scale the distribution profile relative to its maximum value.

Description
Scale the distribution profile relative to its maximum value.

Usage
```r
## S3 method for class 'distrProfile'
scaled(object, session = NULL, what = NULL, ...)
```

Arguments
- `object`: An object of class `distrProfile` as returned by `distributionProfile`.
- `session`: A numeric vector of the sessions to be selected and scaled. Defaults to all sessions.
- `what`: A character version of the variables to be selected and scaled. Defaults to all variables in `object` (what = NULL).
- `...`: Currently not used.

session_duration  
Generic function for calculating session durations

Description
Generic function for calculating session durations

Usage
```r
session_duration(object, session, duration_unit, ...)
```

Arguments
- `object`: The object for which to calculate session durations.
- `session`: The sessions for which to extract sports.
- `duration_unit`: The unit of duration.
- `...`: Arguments to be passed to methods.
**session_times**

Generic function for calculating session times

**Description**

Generic function for calculating session times

**Usage**

```r
session_times(object, session, duration_unit, ...)
```

```r
## S3 method for class 'trackerData'
session_times(object, session = NULL, ...)
```

```r
## S3 method for class 'trackerDataSummary'
session_times(object, session = NULL, ...)
```

**Arguments**

- `object` The object for which to calculate session start and end times.
- `session` The sessions for which to extract sports.
- `duration_unit` The unit durations should be returned.
- `...` Arguments to be passed to methods.

---

**smoother**

Generic function for smoothing

**Description**

Generic function for smoothing

**Usage**

```r
smoother(object, ...)
```

**Arguments**

- `object` The object to be smoothed.
- `...` Arguments to be passed to methods.
**smoother.conProfile**  
*Smoothen for concentration profiles.*

**Description**

To ensure positivity of the smoothed concentration profiles, the concentration profiles are transformed to distribution profiles before smoothing. The smoothed distribution profiles are then transformed to concentration profiles.

**Usage**

```r
## S3 method for class 'conProfile'
smoother(object, session = NULL, what = NULL,
         control = list(...), ...)
```

**Arguments**

- **object**: An object of class `conProfile` as returned by `concentration_profile`.
- **session**: A numeric vector of the sessions to be selected and smoothed. Defaults to all sessions.
- **what**: A character version of the variables to be selected and smoothed. Defaults to all variables in `object` (what = NULL).
- **control**: A list of parameters for controlling the smoothing process. This is passed to `smoother_control.distrProfile`.
- **...**: Arguments to be used to form the default `control` argument if it is not supplied directly.

**See Also**

- `smoother_control.distrProfile`

---

**smoother.distrProfile**  
*Smoothen for distribution profiles.*

**Description**

The distribution profiles are smoothed using a shape constrained additive model with Poisson responses to ensure that the smoothed distribution profile is positive and monotone decreasing.

**Usage**

```r
## S3 method for class 'distrProfile'
smoother(object, session = NULL, what = NULL,
         control = list(...), ...)
```
Arguments

**object**
An object of class `distrProfile` as returned by `distribution_profile`.

**session**
A numeric vector of the sessions to be selected and smoothed. Defaults to all sessions.

**what**
A character version of the variables to be selected and smoothed. Defaults to all variables in `object` (what = `NULL`).

**control**
A list of parameters for controlling the smoothing process. This is passed to `smoother_control.distrProfile`.

**...**
Arguments to be used to form the default control argument if it is not supplied directly.

References


See Also

`smoother_control.distrProfile`

---

**smoother.trackeRdata**  Smoother for `trackeRdata` objects.

**Description**

Smoother for `trackeRdata` objects.

**Usage**

```r
## S3 method for class 'trackeRdata'
smoother(object, session = NULL, control = list(...),
          ...
```

**Arguments**

**object**
An object of class `trackeRdata`.

**session**
The sessions to be smoothed. Default is all sessions.

**control**
A list of parameters for controlling the smoothing process. This is passed to `smoother_control.trackeRdata`.

**...**
Arguments to be used to form the default control argument if it is not supplied directly.
Value

An object of class `trackeRdata`.

See Also

`smoother_control.trackeRdata`

Examples

```r
## Not run:
data('run', package = 'trackeR')
## unsmoothed speeds
plot(run, smooth = FALSE)
## default smoothing
plot(run, smooth = TRUE)
## smoothed with some non-default options
runS <- smoother(run, fun = 'median', width = 20, what = 'speed')
plot(runS, smooth = FALSE)
```

```
smoother_control.distrProfile
Auxiliary function for `smoother.distrProfile`. Typically used to construct a control argument for `smoother.distrProfile`.

Description

Auxiliary function for `smoother.distrProfile`. Typically used to construct a control argument for `smoother.distrProfile`.

Usage

```r
smoother_control.distrProfile(k = 30, sp = NULL, parallel = FALSE)
```

```r
smootherControl.distrProfile(k = 30, sp = NULL, parallel = FALSE)
```

Arguments

- `k` Number of knots.
- `sp` A vector of smoothing parameters passed on to `scam`.
- `parallel` Logical. Should computation be carried out in parallel?
**smoother_control.trackerdata**

*Auxiliary function for smoother.trackerdata. Typically used to construct a control argument for smoother.trackerdata.*

---

**Description**

Auxiliary function for `smoother.trackerdata`. Typically used to construct a control argument for `smoother.trackerdata`.

**Usage**

```r
smoother_control.trackerdata(fun = "mean", width = 10, parallel = FALSE, what = c("speed", "heart_rate"), nsessions = NA, ...)
```

```r
smootherControl.trackerdata(fun = "mean", width = 10, parallel = FALSE, what = c("speed", "heart_rate"), nsessions = NA, ...)
```

**Arguments**

- **fun**: The name of the function to be matched and used to aggregate/smooth the data.
- **width**: The width of the window in which the raw observations get aggregated via function `fun`.
- **parallel**: Logical. Should computation be carried out in parallel? If `TRUE` computation is performed in parallel using the backend provided to `foreach`. Default is `FALSE`.
- **what**: Vector of the names of the variables which should be smoothed.
- **nsessions**: Vector containing the number of session. Default corresponds to all sessions belonging to the same group. Used only internally.
- **...**: Currently not used.

**See Also**

`smoother.trackerdata`

---

**sort.trackerdata**

*Sort sessions in trackerData objects*

**Description**

Sort the sessions `trackerData` objects into ascending or descending order according to the first session timestamp.
summary.trackerdata

Usage

## S3 method for class 'trackerdata'
sort(x, decreasing = FALSE, ...)

Arguments

- `x`: A trackerdata object.
- `decreasing`: Logical. Should the objects be sorted in increasing or decreasing order?
- `...`: Currently not used.

Description

Convert speed to distance.

Usage

speed2distance(speed, time, timeunit, cumulative = TRUE)

Arguments

- `speed`: Speed in meters per second.
- `time`: Time.
- `timeunit`: Time unit in speed, e.g., "hours" for speed in *_per_h.
- `cumulative`: Logical. Should the cumulative distances be returned?

Value

Distance in meters.

summary.trackerdata

Summary of training sessions

Description

Summary of training sessions

Usage

## S3 method for class 'trackerRdata'
summary(object, session = NULL,
         moving_threshold = NULL, unit_reference_sport = NULL, ...)


Arguments

object     An object of class `trackeRdata`.
session    A numeric vector of the sessions to be summarised, defaults to all sessions.
moving_threshold
           A named vector of 3 speeds above which an athlete is considered moving, given in the unit of the speed measurements in `object`. If NULL (default), the speeds are taken to be `c(cycling = 2, running = 1, swimming = 0.5)`. See Details.
unit_reference_sport
           The sport to inherit units from (default is taken to be the most frequent sport in `object`).

Details

The default speed thresholds are 1 m/s for running (3.6 km/h; slow walking), 2 m/s for cycling (7.2 km/h) for cycling and 0.5 m/s (1.8km/h) for swimming. For reference, the preferred walking speed for humans is around 1.4 m/s (Bohannon, 1997).

The units for the computed summaries match those of the sport specified by `unit_reference_sport`.

If `object` has thresholds then the thresholds that match those of the sport specified by `unit_reference_sport` are applied to the respective summaries.

Value

An object of class `trackeRdataSummary`.

References


See Also

`plot.trackeRdataSummary`

Examples

data('runs', package = 'trackeR')
runSummary <- summary(runs, session = 1:2)
## print summary
runSummary
print(runSummary, digits = 3)
## Not run:
## change units
change_units(runSummary, variable = 'speed', unit = 'km_per_h')
## plot summary
runSummaryFull <- summary(runs)
plot(runSummaryFull)
plot(runSummaryFull, group = c('total', 'moving'),

plot(trackeRdata)
63
what = c('avgSpeed', 'distance', 'duration', 'avgHeartRate'))

## End(Not run)

---

**threshold.trackerdata**  
Thresholding for variables in trackerdata objects

**Description**

Thresholding for variables in trackerdata objects

**Usage**

```r
## S3 method for class 'trackerdata'
threshold(object, variable, lower, upper, sport, ...)

threshold(object, ...)
```

**Arguments**

- `object`  
  An object of class `trackerdata`.

- `variable`  
  A vector containing the names of the variables to which thresholding is applied. See Details.

- `lower`  
  A vector containing the corresponding lower thresholds. See Details.

- `upper`  
  A vector containing the corresponding upper thresholds. See Details.

- `sport`  
  A vector of sports (amongst 'cycling', 'running', 'swimming') with each element corresponding to `variable`, `lower` and `upper`.

- `...`  
  Currently not used.

**Details**

`lower` and `upper` are always understood as referring to the units of the object.

If the arguments `variable`, `lower`, and `upper` are all unspecified, the following default thresholds are employed:

- latitude [-90, 90] degrees
- longitude [-180, 180] degrees
- altitude [-500, 9000] m
- distance [0, Inf] meters
- cadence_running [0, Inf] steps per min
- cadence_cycling [0, Inf] revolutions per min
- speed [0, Inf] meters
- heart rate [0, 250] bpm
- power [0, Inf] W
timeAboveThreshold

• pace [0, Inf] min per km
• duration [0, Inf] seconds
• temperature [-20, 60] C

after they have been tranformed to the units of the object

The thresholds for speed differ across sports: for running they are [0, 12.5] meters per second, for cycling [0, 100] meters per second and for swimming [0, 5] meters per second.

Examples

```r
## Not run:
data('runs', package = 'trackeR')
plot(runs, session = 4, what = 'speed', threshold = FALSE)
runsT <- threshold(runs, variable = 'speed', lower = 0, upper = 12.5, sport = "running")
plot(runsT, session = 4, what = 'speed', threshold = FALSE)
## End(Not run)
```

timeAboveThreshold Time spent above a certain threshold.

Description

Time spent above a certain threshold.

Usage

```
timeAboveThreshold(object, threshold = 1, ge = TRUE)
```

Arguments

- object: A (univariate) zoo object.
- threshold: The threshold.
- ge: Logical. Should time include the threshold (greater or equal to threshold) or not (greater only)?
Generic function for visualising the sessions on a time versus date plot

Description

Generic function for visualising the sessions on a time versus date plot

Timeline plot for \texttt{trackerdata} objects.

Timeline plot for \texttt{trackerdataSummary} objects

Usage

timeline(object, lims, ...)

## S3 method for class 'trackerdata'
timeline(object, lims = NULL, ...)

## S3 method for class 'trackerdataSummary'
timeline(object, lims = NULL, ...)

Arguments

- \texttt{object} \hspace{1cm} An object of class \texttt{trackerdata} or \texttt{trackerdataSummary}.
- \texttt{lims} \hspace{1cm} An optional vector of two times in HH:MM format. Default is NULL. If supplied, the times are used to define the limits of the time axis.
- \texttt{...} \hspace{1cm} Arguments passed to \texttt{summary.trackerdata}.

Examples

```r
## Not run:
data('runs', package = 'trackeR')
## timeline plot applied on the \code{trackerdata} object directly and with
## inferred limits for the time axis
timeline(runs)

## the same timeline plot applied on the \code{trackerdataSummary} object
runSummary <- summary(runs)
timeline(runSummary, lims = c('00:01', '23:59'))

## End(Not run)
```
trackeR: Infrastructure for running and cycling data from GPS-enabled tracking devices

Description

trackeR provides infrastructure for handling cycling and running data from GPS-enabled tracking devices. After extraction and appropriate manipulation of the training or competition attributes, the data are placed into session-aware data objects with an S3 class trackeRdata. The information in the resultant data objects can then be visualised, summarised and analysed through corresponding flexible and extensible methods.

Note

Core facilities in the trackeR package, including reading functions (see readX), data pre-processing strategies (see trackeRdata), and calculation of concentration and distribution profiles (see distributionProfile and concentrationProfile) are based on un-packaged R code that was developed by Ioannis Kosmidis for the requirements of the analyses in Kosmidis & Passfield (2015).

Note

This work has been supported by the English Institute of Sport http://www.eis2win.co.uk and University College London (UCL), which jointly contributed to the grant that funded Hannah Frick’s Post Doctoral Research Fellowship at UCL between 2014 and 2016 and a percentage of Ioannis Kosmidis’ time. Ioannis Kosmidis has also been supported by the Alan Turing Institute under the EPSRC grant EP/N510129/1 (Turing award number TU/B/000082). The support of the aforementioned organisations is greatly acknowledged.

Hannah Frick maintained trackeR from its first release up and since version 1.0.0.

References


trackeRdata

Create a trackeRdata object

Description

Create a trackeRdata object from a data frame with observations being divided in separate training sessions. For breaks within a session observations are imputed.
trackeRdata

Usage

trackeRdata(dat, units = NULL, sport = NULL, session_threshold = 2,
correct_distances = FALSE, from_distances = TRUE, country = NULL,
mask = TRUE, lgap = 30, lskip = 5, m = 11, silent = FALSE)

Arguments

dat A data.frame object.
units The output of generate_units.
sport What sport does dat contain data of? Either 'cycling', 'running', 'swimming' or NULL (default), in which case the sport is directly extracted from the dat. See Details.
session_threshold The threshold in hours for the time difference between consecutive timestamps above which they are considered to belong to different training sessions.
correct_distances Logical. Should the distances be corrected for elevation?
from_distances Logical. Should the speeds be calculated from the distance recordings instead of taken from the speed recordings directly?
country ISO3 country code for downloading altitude data. If NULL, country is derived from longitude and latitude
mask Logical. Passed on to getData. Should only the altitudes for the specified country be extracted (TRUE) or also those for the neighboring countries (FALSE)?
lgap Time in seconds corresponding to the minimal sampling rate.
lskip Time in seconds between the last observation before a small break and the first imputed speed or the last imputed speed and the first observation after a small break.
m Number of imputed observations in each small break.
silent Logical. Should warnings be generated if any of the sanity checks on the data are triggered?

Details

During small breaks within a session, e.g., because the recording device was paused, observations are imputed the following way: 0 for speed, last known position for latitude, longitude and altitude, NA or 0 power for running or cycling session, respectively, and NA for all other variables. Distances are (re-)calculated based on speeds after imputation.

trackeRdata assumes that all observations in dat are from the same sport, even if dat ends up having observations from different sessions (also depending on the value of session_threshold). if attr(dat, 'sport') is NA then the current implementation of trackeRdata returns an error.

References

See Also

`readContainer` for reading .tcx and .db3 files directly into trackerRdata objects.

Examples

```r
# read raw data
filepath <- system.file('extdata/tcx/', '2013-06-08-090442.TCX', package = 'trackerR')
run0 <- readTCX(file = filepath, timezone = 'GMT')

# turn into trackerRdata object
units0 <- generate_units()
run0 <- trackerRdata(run0, units = units0)
```

---

**unique.trackerRdata**  
**Extract unique sessions in a trackerRdata object**

Description

Extract unique sessions in a trackerRdata object

Usage

```r
# S3 method for class 'trackerRdata'
unique(x, incomparables = FALSE, ...)
```

Arguments

- `x`  
  A trackerRdata object.

- `incomparables`  
  Currently not used.

- `...`  
  Currently not used.

Details

Uniqueness is determined by comparing the first timestamp of the sessions in the trackerRdata object.
**Description**

Calculate $W'$ expended, i.e., the work capacity above critical power/speed which has been depleted and not yet been replenished.

**Usage**

$\text{Wexp}(\text{object, w0, cp, version} = \text{c("2015", "2012"), meanRecoveryPower} = \text{FALSE})$

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Univariate <code>zoo</code> object containing the time stamped power output or speed values. (Power should be in Watts, speed in meters per second.)</td>
</tr>
<tr>
<td>w0</td>
<td>Initial capacity of $W'$, as calculated based on the critical power model by Monod and Scherrer (1965).</td>
</tr>
<tr>
<td>cp</td>
<td>Critical power/speed, i.e., the power/speed which can be maintained for longer period of time.</td>
</tr>
<tr>
<td>version</td>
<td>How should $W'$ be replenished? Options include ‘2015’ and ‘2012’ for the versions presented in Skiba et al. (2015) and Skiba et al. (2012), respectively. See Details.</td>
</tr>
<tr>
<td>meanRecoveryPower</td>
<td>Should the mean of all power outputs below critical power be used as recovery power? See Details.</td>
</tr>
</tbody>
</table>

**Details**

Skiba et al. (2015) and Skiba et al. (2012) both describe an exponential decay of $W'$ expended over an interval $[t_{i-1}, t_i)$ if the power output during this interval is below critical power:

$$W_{\text{exp}}(t_i) = W_{\text{exp}}(t_{i-1}) \times \exp(nu \times (t_i - t_{i-1}))$$

However, the factor $nu$ differs: Skiba et al. (2012) describe it as $1/\tau$ with $\tau$ estimated as

$$\tau = 546 \times \exp(-0.01 \times (CP - P_i)) + 316$$

Skiba et al. (2015) use $(P_i - CP)/W'_0$. Skiba et al. (2012) and Skiba et al. (2015) employ a constant recovery power (calculated as the mean over all power outputs below critical power). This rationale can be applied by setting the argument `meanRecoveryPower` to TRUE. Note that this uses information from all observations with a power output below critical power, not just those prior to the current time point.
**References**


---

**Wprime**

\[ W' \]: work capacity above critical power/speed.

**Description**

\[ W' \]: work capacity above critical power/speed.

**Usage**

\[
Wprime(object, session = NULL, quantity = c("expended", "balance"), w0, cp, version = c("2015", "2012"), meanRecoveryPower = FALSE, parallel = FALSE, ...)
\]

**Arguments**

- **object**: A `trackeRdata` object.
- **session**: A numeric vector of the sessions to be used, defaults to all sessions.
- **quantity**: Should \( W' \) 'expended' or \( W' \) 'balance' be returned?
- **w0**: Initial capacity of \( W' \), as calculated based on the critical power model by Monod and Scherrer (1965).
- **cp**: Critical power/speed, i.e., the power/speed which can be maintained for longer period of time.
- **version**: How should \( W' \) be replenished? Options include '2015' and '2012' for the versions presented in Skiba et al. (2015) and Skiba et al. (2012), respectively. See Details.
- **meanRecoveryPower**: Should the mean of all power outputs below critical power be used as recovery power? See Details.
- **parallel**: Logical. Should computation be carried out in parallel? If TRUE computation is performed in parallel using the backend provided to `foreach`. Default is FALSE.
- **...**: Currently not used.
Details

# Skiba et al. (2015) and Skiba et al. (2012) both describe an exponential decay of $W'$ expended over an interval $[t_{i-1}, t_i)$ if the power output during this interval is below critical power:

$$W_{\text{exp}}(t_i) = W_{\text{exp}}(t_{i-1}) \times \exp(nu \times (t_i - t_{i-1}))$$

However, the factor $nu$ differs: Skiba et al. (2012) describe it as $1/\tau$ with $\tau$ estimated as

$$\tau = 546 \times \exp(-0.01 \times (CP - P_i)) + 316$$

Skiba et al. (2015) use $(P_i - CP)/W_{0}^{'}$. Skiba et al. (2012) and Skiba et al. (2015) employ a constant recovery power (calculated as the mean over all power outputs below critical power). This rationale can be applied by setting the argument `meanRecoveryPower` to TRUE. Note that this uses information from all observations with a power output below critical power, not just those prior to the current time point.

Value

An object of class `trackRWprime`.

References


Examples

```r
## Not run:
data('runs', package = 'trackerR')
wexp <- Wprime(runs, session = c(11,13), cp = 4, version = '2012')
plot(wexp)

## End(Not run)
```
zones

Time spent in training zones.

Description

Time spent in training zones.

Usage

zones(object, session = NULL, what = c("speed", "heart_rate"),
       breaks = NULL, parallel = FALSE, n_zones = 9,
       unit_reference_sport = NULL, ...)

Arguments

object An object of class trackerRdata.
session A numeric vector of the sessions to be plotted, defaults to all sessions.
what A vector of variable names.
breaks A list of breakpoints between zones, corresponding to the variables in what.
parallel Logical. Should computation be carried out in parallel? If TRUE computation is
performed in parallel using the backend provided to foreach. Default is FALSE.
n_zones numeric that sets the number of zones for data to be split into. Default is 9.
unit_reference_sport The sport to inherit units from (default is taken to be the most frequent sport in
object).
...
... Currently not used.

Value

An object of class trackerRdataZones.

See Also

plot.trackeRdataZones

Examples

data('run', package = 'trackeR')
runZones <- zones(run, what = 'speed', breaks = list(speed = c(0, 2:6, 12.5)))
# if breaks is a named list, argument 'what' can be left unspecified
runZones <- zones(run, breaks = list(speed = c(0, 2:6, 12.5)))
# if only a single variable is to be evaluated, 'breaks' can also be a vector
runZones <- zones(run, what = 'speed', breaks = c(0, 2:6, 12.5))
plot(runZones)
Index

*Topic datasets
  run, 54
  runs, 54
  1965), (Wprime), 71
  2012). (Wprime), 71
  above (Wprime), 71
  again, (Wprime), 71
  al., (Wprime), 71
  and (Wprime), 71
  append, 4
  append.trackRdata, 4
  applied (Wprime), 71
  as.POSIXct, 47, 48, 50
  available, (Wprime), 71
  balance. (Wprime), 71
  Based (Wprime), 71
  been (Wprime), 71
  below (Wprime), 71
  bpm2bpm (conversions), 13
  by (Wprime), 71
  C2C (conversions), 13
  c2d, 5
  C2F (conversions), 13
  capacity (Wprime), 71
  change_units, 5
  change_units.conProfile, 6
  change_units.distrProfile, 6
  change_units.trackRdata, 7
  change_units.trackRdataSummary, 7
  change_units.trackRdataZones, 8
  change_units.trackRwPrime, 8
  changeUnits (change_units), 5
  collect_units, 9
  compute_breaks, 9
  compute_limits, 9, 10
  concentration_profile, 10, 29, 37, 52, 58
  concentration_profile.distrProfile, 11
  concentration_profile.trackRdata
    (concentration_profile.distrProfile), 11
  concentrationProfile, 6, 27, 46, 67
  concentrationProfile
    (concentration_profile), 10
  conProfile, 6, 21
  conProfile
    (concentration_profile.distrProfile), 11
  conversions, 13
  critical (Wprime), 71
  cycling (Wprime), 71
  data.frame, 43, 44, 68
  Data2fd, 46
  decreasing_smother, 17
  decreasing_Smooth
    (decreasing_smother), 17
  degree2degree (conversions), 13
  depleted (Wprime), 71
  describes (Wprime), 71
  distance2speed, 18
  distribution_profile, 19, 29, 37, 53, 59
  distributionProfile, 6, 28, 46, 56, 67
  distributionProfile
    (distribution_profile), 19
  distrProfile, 11, 12
  distrProfile(distribution_profile), 19
  during (Wprime), 71
  et (Wprime), 71
  exercise (Wprime), 71
  expended, (Wprime), 71
  F2C (conversions), 13
  F2F (conversions), 13
  fd, 46
  find_unit_reference_sport, 20
  finite (Wprime), 71
foreach (Wprime), 71
foreach, 51, 61, 71, 73
fortify_conProfile, 21
fortify_distrProfile, 21
fortify_trackerRdata, 22
fortify_trackerRdataSummary, 22
fortify_trackerRWprime, 23
fortify_conProfile
  (fortify_conProfile), 21
fortify_distrProfile
  (fortify_distrProfile), 21
fortify_trackerRdata
  (fortify_trackerRdata), 22
fortify_trackerRdataSummary
  (fortify_trackerRdataSummary, 22
fortify_trackerRWprime
  (fortify_trackerRWprime), 23
ft2ft (conversions), 13
ft2km (conversions), 13
ft2mi (conversions), 13
ft_per_min2ft_per_min (conversions), 13
ft_per_min2ft_per_s (conversions), 13
ft_per_min2km_per_h (conversions), 13
ft_per_min2km_per_min (conversions), 13
ft_per_min2m_per_s (conversions), 13
ft_per_min2mi_per_h (conversions), 13
ft_per_min2mi_per_min (conversions), 13
ft_per_s2ft_per_min (conversions), 13
ft_per_s2ft_per_s (conversions), 13
ft_per_s2km_per_h (conversions), 13
ft_per_s2km_per_min (conversions), 13
ft_per_s2m_per_s (conversions), 13
ft_per_s2mi_per_h (conversions), 13
ft_per_s2mi_per_min (conversions), 13
funPCA, 23, 41
GC2trackeRdata, 24
generate_thresholds, 25
generate_units, 9, 26, 44, 68
generateBaseUnits (generate_units), 26
generateDefaultThresholds
  (generate_thresholds), 25
get_map, 43
get_operations, 27
get_operations_conProfile, 27
get_operations_distrProfile, 12, 19, 28
generate_thresholds (Wprime), 12, 20, 27
get_profile (get_profile_distrProfile), 29
get_profile_distrProfile, 29
generate_default_thresholds (Wprime), 27
get_resting_periods, 29
generate_default_thresholds (Wprime), 30
generate_default_thresholds (Wprime), 30
get_units, 12, 20, 31
generate_default_thresholds (Wprime), 31
generate_default_thresholds (Wprime), 32
generate_default_thresholds (Wprime), 32
generate_default_thresholds (Wprime), 33
generate_default_thresholds (Wprime), 33
generate_default_thresholds (Wprime), 34
get_units_distrProfile, 25, 49, 50, 68
generate_default_thresholds (Wprime), 27
generate_default_thresholds (Wprime), 31
generate_default_thresholds (Wprime), 43
h2h (conversions), 13
h2min (conversions), 13
h2s (conversions), 13
h_per_km2min_per_km (conversions), 13
h_per_km2min_per_mi (conversions), 13
h_per_km2min_per_km (conversions), 13
h_per_km2min_per_km (conversions), 13
has (Wprime), 71
how (Wprime), 71
impute_speeds, 34
impute_speeds (impute_speeds), 34
interest (Wprime), 71
is (Wprime), 71
it (Wprime), 71
km2ft (conversions), 13
km2km (conversions), 13
km2m (conversions), 13
km2mi (conversions), 13
km_per_h2ft_per_min (conversions), 13
km_per_h2ft_per_s (conversions), 13
km_per_h2km_per_h (conversions), 13
km_per_h2km_per_min (conversions), 13
km_per_h2km_per_min (conversions), 13
km_per_h2km_per_min (conversions), 13
km_per_h2mi_per_min (conversions), 13
km_per_h2mi_per_min (conversions), 13
km_per_h2mi_per_min (conversions), 13
km_per_h2mi_per_min (conversions), 13
km_per_min2ft_per_min (conversions), 13
km_per_min2ft_per_s (conversions), 13
km_per_min2ft_per_s (conversions), 13
km_per_min2ft_per_s (conversions), 13
km_per_min2km_per_h (conversions), 13
km_per_min2km_per_h (conversions), 13
km_per_min2km_per_h (conversions), 13
km_per_min2km_per_h (conversions), 13
km_per_min2km_per_h (conversions), 13
gfmap, 43
get_operations_distrProfile, 12, 19, 28
INDEX

readX, 47, 49, 67
replenished (Wprime), 71
replenished (Wprime), 71
respectively (Wprime), 71
restingPeriods (get_resting_periods), 29
rev_per_min (conversions), 13
rev_per_min2 (conversions), 13
ridges, 51
ridges.connProfile, 52
ridges.distrProfile, 53
ridges.trackRdata, 53
run, 54
runners (Wprime), 71
runs, 54
s2h (conversions), 13
s2min (conversions), 13
s2s (conversions), 13
s_per_m2min_per_km (conversions), 13
s_per_m2min_per_mi (conversions), 13
s_per_m2s_per_m (conversions), 13
sanity_checks, 55
scaled, 55
scaled.distrProfile, 56
scam, 18, 60
Scherrer, (Wprime), 71
session_duration, 56
session_times, 57
smoother, 57
smoother.connProfile, 58
smoother.distrProfile, 58, 60
smoother.trackRdata, 59, 61
smoother_control.distrProfile, 37, 52, 53, 58, 59, 60
smoother_control.trackRdata, 59, 60, 61
smootherControl.distrProfile
(smoother_control.distrProfile), 60
smootherControl.trackRdata, 38
smootherControl.trackRdata
(smoother_control.trackRdata), 61
sort.trackRdata, 61
speed (Wprime), 71
speed, (Wprime), 71
speed2distance, 62
steps_per_min (conversions), 13
steps_per_min2 (conversions), 13
still (Wprime), 71
substituting (Wprime), 71
summary, 45
summary.trackRdata, 40, 62, 66
the (Wprime), 71
This (Wprime), 71
this (Wprime), 71
threshold, 35, 38, 43
threshold (threshold.trackRdata), 64
threshold.trackRdata, 64
Thus, (Wprime), 71
timeAboveThreshold, 65
timeline, 66
to (Wprime), 71
trackR, 67
trackR-package (trackR), 67
trackRdata, 7, 9–12, 19, 20, 22, 25, 28, 32, 35, 38, 43, 45, 49, 51, 54, 59–61, 63, 64, 66, 67, 67, 71, 73
trackRdataSummary, 39, 66
trackRdataSummary
(summary.trackRdata), 62
trackRWprime, 8
trackRWprime (Wprime), 71

unique.trackRdata, 69

W (Wprime), 71
W' (Wprime), 71
W2kW (conversions), 13
W2W (conversions), 13
Wexp, 70
While (Wprime), 71
with (Wprime), 71
work (Wprime), 71
Wprime, 23, 42, 71

yet (Wprime), 71
zones, 40, 73
zoo, 34, 35, 70