Package ‘trackeR’

May 15, 2019

Version 1.5.2

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, gttable

Suggests testthat, knitr, rmarkdown, covr

VignetteBuilder knitr

License GPL-3

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

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

RoxygenNote 6.1.1

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 2019-05-15 11:50:03 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 ................................... 9
collect_units .................................................... 9
compute_breaks ................................................ 10
compute_limits ............................................... 10
concentration_profile ......................................... 11
concentration_profile.distrProfile ............................ 12
conversions ...................................................... 13
decreasing_smoother .......................................... 18
distance2speed ................................................ 19
distribution_profile .......................................... 19
find_unit_reference_sport ..................................... 21
fortify.conProfile ............................................ 21
fortify.distrProfile ........................................... 22
fortify.trackeRdata .......................................... 22
fortify.trackeRdataSummary .................................. 23
fortify.trackeRWprime ....................................... 23
funPCA .......................................................... 24
GC2trackeRdata ............................................... 25
generate_thresholds ......................................... 26
generate_units ............................................... 26
get_elevation_gain .......................................... 27
get_operations ............................................... 28
get_operations.conProfile ..................................... 28
get_operations.distrProfile .................................... 29
get_operations.trackeRdata ................................... 29
get_profile.distrProfile ..................................... 30
get_resting_periods ......................................... 30
get_sport.trackeRWprime .................................... 31
get_units ...................................................... 32
get_units.conProfile ......................................... 32
get_units.distrProfile ....................................... 33
get_units.trackeRdata ....................................... 33
get_units.trackeRdataSummary ............................... 34
get_units.trackeRdataZones .................................. 34
get_units.trackeRWprime .................................... 35
impute_speeds ............................................... 35
leaflet_route .................................................. 36
nsessions.tracRprime ........................................... 37
plot.conProfile ................................................... 37
plot.distrProfile .................................................. 38
plot.tracRdata .................................................... 39
plot.tracRdataSummary .......................................... 40
plot.tracRdataZones ............................................. 41
plot.tracRfpca .................................................... 42
plot.tracRprime ................................................... 43
plot_route ......................................................... 44
prepare_route ..................................................... 45
prettifyUnit ....................................................... 45
print.tracRdata .................................................... 46
print.tracRdataSummary ......................................... 47
profile2fd ........................................................ 47
readX ............................................................. 48
read_container ................................................... 49
read_directory ..................................................... 51
ridges .............................................................. 53
ridges.conProfile ................................................ 54
ridges.distrProfile .............................................. 55
ridges.tracRdata .................................................. 55
run ................................................................. 56
runs ................................................................. 56
sanity_checks ..................................................... 57
scaled .............................................................. 57
scaled.distrProfile ............................................... 58
session_duration .................................................. 58
session_times ...................................................... 59
smoother ........................................................ 59
smoother.conProfile ............................................. 60
smoother.distrProfile .......................................... 60
smoother.tracRdata .............................................. 61
smoother_control.distrProfile .................................. 62
smoother_control.tracRdata .................................... 63
sort.tracRdata ...................................................... 63
speed2distance .................................................... 64
summary.tracRdata .............................................. 64
threshold.tracRdata ............................................. 66
timeAboveThreshold ............................................. 67
timeline ........................................................... 68
tracR ............................................................. 69
tracRdata ........................................................ 69
unique.tracRdata ............................................... 71
Wexp ............................................................. 72
Wprime ........................................................... 73
zones ............................................................. 75

Index ............................................................ 76
**append**  
*Generic function for appending data to existing files*

**Description**

Generic function for appending data to existing files

**Usage**

```r
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**

```r
## 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.
Transform concentration profile to distribution profile.

Usage

c2d(cp)

Arguments

- **cp**: Single concentration profile as a zoo object.

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

```r
## 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

```r
## 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.
Description

Change the units of the variables in an 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.

Description

Change the units of the variables in an trackerthresholds object

Usage

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

Arguments

- `object` An object of class trackerthresholds.
- `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 the units of the variables in an `trackeRWprime` object

**Description**

Change the units of the variables in an `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**  
```r  
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**  
```r  
compute_limits(object, a = 1e-04)  
```
**concentration_profile**

Arguments

- **object**: A `trackerdata` 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

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

Arguments

- **object**: An object of class `trackerRdata` 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 = 'trackerR')
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 'trackerRdata' object, which is a
```
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, ...)
```

```r
## 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).
- **limits**: A named list of vectors of two numbers to specify the lower 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 com-
ponent 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 trackRdata object, after inheriting units from unit_reference_sport.

References


conversions 13

Auxiliary conversion functions

Description

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

Usage

m2km(variable)

km2m(variable)

m2ft(variable)

ft2m(variable)
m2mi(variable)
mi2m(variable)
kmi2ft(variable)
ft2km(variable)
km2mi(variable)
mi2km(variable)
ft2mi(variable)
mi2ft(variable)
m2m(variable)
kmi2km(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)
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)
conversions

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_min\textsubscript{2} \text{mi\_per\_min}(\text{variable})

bpm\textsubscript{2} \text{bpm}(\text{variable})

s\textsubscript{2} \text{min\_per\_km}(\text{variable})

min\textsubscript{2} \text{min\_per\_m}(\text{variable})

s\textsubscript{2} \text{min\_per\_mi}(\text{variable})

min\textsubscript{2} \text{min\_per\_mi}(\text{variable})

min\textsubscript{2} \text{min\_per\_km}(\text{variable})

min\textsubscript{2} \text{min\_per\_km}(\text{variable})

min\textsubscript{2} \text{min\_per\_mi}(\text{variable})

min\textsubscript{2} \text{min\_per\_mi}(\text{variable})

h\textsubscript{2} \text{min\_per\_km}(\text{variable})

h\textsubscript{2} \text{min\_per\_km}(\text{variable})

h\textsubscript{2} \text{min\_per\_mi}(\text{variable})

h\textsubscript{2} \text{min\_per\_mi}(\text{variable})

W\textsubscript{2} \text{kW}(\text{variable})

kW\textsubscript{2} \text{W}(\text{variable})

W\textsubscript{2} \text{W}(\text{variable})

kW\textsubscript{2} \text{kW}(\text{variable})

steps\textsubscript{2} \text{steps\_per\_min}(\text{variable})

rev\textsubscript{2} \text{rev\_per\_min}(\text{variable})

steps\textsubscript{2} \text{steps\_per\_min}(\text{variable})

rev\textsubscript{2} \text{rev\_per\_min}(\text{variable})
decreasing_smoother

\[ \text{rev\_per\_min2steps\_per\_min}(\text{variable}) \]
\[ \text{C2F}(\text{variable}) \]
\[ \text{C2C}(\text{variable}) \]
\[ \text{F2F}(\text{variable}) \]
\[ \text{F2C}(\text{variable}) \]

**Arguments**

| variable | Variable to be converted. |

**Description**

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

**Usage**

\[ \text{decreasing\_smoother}(x, y, k = 30, \text{len} = \text{NULL}, \text{sp} = \text{NULL}) \]
\[ \text{decreasingSmooother}(x, y, k = 30, \text{len} = \text{NULL}, \text{sp} = \text{NULL}) \]

**Arguments**

<table>
<thead>
<tr>
<th>x</th>
<th>The regressor passed on to the formula argument of scam.</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>The response passed on to the formula argument of scam.</td>
</tr>
<tr>
<td>k</td>
<td>Number of knots.</td>
</tr>
<tr>
<td>len</td>
<td>If NULL, the default, x is used for prediction. Otherwise, prediction is done over the range of x with len equidistant points.</td>
</tr>
<tr>
<td>sp</td>
<td>A vector of smoothing parameters passed on to scam.</td>
</tr>
</tbody>
</table>
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. This is the output of get_units on the corresponding trackerdata object, after inheriting units from unit_reference_sport.

References


Examples

data('run', package = 'trackeR')
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

find_unit_reference_sport(object)

Arguments

object any object with a get_sport method implemented (run methods(get_sport)).

fortify_conProfile Fortify a conProfile object for plotting with ggplot2.

Description

Fortify a conProfile object for plotting with ggplot2.

Usage

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

fortify_conProfile(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.distrProfile

Fortify a distrProfile object for plotting with ggplot2.

Description

Fortify a distrProfile object for plotting with ggplot2.

Usage

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

fortify_distrProfile(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.trackeRdata

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, ...)

fortify_trackerdata(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.trackeRdataSummary**

*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, ...)

fortify_trackeRdataSummary(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 `trackRfpca`.

**References**


**Examples**

```r
## Not run:
data('runs', package = 'trackR')
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
```
GC2trackeRdata

Coercion function for use in Golden Cheetah

Description

Coercion function for use in Golden Cheetah

Usage

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

Arguments

gc Output of GC.activity.
cycling Logical. Does the data stem from cycling?
correct_distances Logical. Should the distances be corrected for elevation? Default is FALSE.
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?
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?

See Also

trackeRdata
**generate_thresholds**

*Generate default thresholds.*

**Description**
Generate default thresholds.

**Usage**

```r
generate_thresholds(variable, lower, upper, sport, ...)
generateDefaultThresholds(variable, lower, upper, sport, ...)
```

**Arguments**

- `variable` A vector of variables with user-specified thresholds.
- `lower` A vector of lower limits corresponding to the elements of `variable`.
- `upper` A vector of upper limits corresponding to the elements of `variable`.
- `sport` A vector of sports (amongst 'cycling', 'running', 'swimming') with each element corresponding to `variable`, `lower` and `upper`.
- `...` Currently not used.

**generate_units**

*Generate and set base units.*

**Description**
Generate and set base units.

**Usage**

```r
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.
get_elevation_gain

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_elevation_gain  (Cumulative) Elevation gain.

Description

(Cumulative) Elevation gain.

Usage

get_elevation_gain(object, smooth = FALSE, cumulative = FALSE, vertical_noise = 0)

Arguments

- object: A (univariate) zoo object.
- smooth: Logical. Should the elevation be smoothed? Default is TRUE.
- cumulative: Logical. Return the cumulative elevation gain (FALSE; default) or just the elevation gain?
- vertical_noise: A scalar. Absolute elevation gains less that vertical_noise are set to zero. Default is 0.

Details

The elevation gain is defined here as the difference in altitude between two consecutive observations. If cumulative = FALSE then the elevation gain is returned, otherwise any elevation loses (i.e. negative elevation gain) are ignored and the cumulative elevation gain is returned. If smooth = TRUE then the elevation gain will be smoothed using a spline smoother before either returning it or computing cumulative elevation gains.
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.

---

g_{\text{et\_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**

```r
## 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**

```r
## 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

```r
## 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

```r
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, ...)

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

get_sport(object, session, ...)

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

## S3 method for class 'trackeRdataSummary'
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

Generic function for extracting the units of measurement

Description

Generic function for extracting the units of measurement

Usage

get_units(object, ...)

getUnits(object, ...)

Arguments

object The object of which the units of measurement are retrieved.
...

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

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

Arguments

object An object of class conProfile.
...

Currently not used.
get_units.distrProfile

*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_units.trackerdata  *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_unitstrackerdataZones**

*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_unitstrackerdataZones**

*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 an trackeRWprime object

Description

Get the units of the variables in an 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)
impute_speeds(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.
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://maps.com

Usage

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

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

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

## End(Not run)
nsessions.trackeRWprime

**Generic function for calculating number of sessions**

**Description**

Generic function for calculating number of sessions

**Usage**

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

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

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

```r
## S3 method for class 'trackeRdataSummary'
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? A vector with at least one of "latitude", "longitude", "altitude", "distance", "heart_rate", "speed", "cadence_running", "cadence_cycling", "power", "temperature", "pace", "cumulative_elevation_gain". Default is c("pace", "heart_rate").
- `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`.

See Also

trackeRdata

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)
#
## Speed and elevation gain
plot(runs, session = 2:10, what = c("speed", "cumulative_elevation_gain"), trend = FALSE)
## End(Not run)
```

---

`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. A vector with at least one of "distance", "duration", "avgSpeed", "avgPace", "avgCadenceRunning", "avgCadenceCycling", "avgAltitude", "avgPower", "avgHeartRate", "avgTemperature", "wRRatio", "total_elevation_gain", and NULL, in which case all variables are plotted.
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.

See Also

summary.trackerData

Examples

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

## End(Not run)

plot.trackerDataZones  Plot training zones.

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

```r
## Not run:
data('run', package = 'trackeR')
runZones <- zones(run, what = 'speed', breaks = c(0, 2:6, 12.5))
plot(runZones, percent = FALSE)

## 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`
plot.trackerWprime

Examples

## Not run:
data('runs', package = 'trackerR')
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.trackerWprime  
Plot W'.

Description

Plot W'.

Usage

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

Arguments

x  
An object of class trackerWprime 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

## Not run:
data('runs', package = 'trackerR')
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, maptype = "toner", messaging = FALSE, ...)

plotRoute(x, session = 1, zoom = NULL, speed = TRUE, threshold = TRUE, mfrow = NULL, maptype = "toner", messaging = FALSE, ...)
```

**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_stamenmap` (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. # @param source Passed to `get_map`. Default is "stamen".
- `maptype` Passed to `get_stamenmap`. Default is "toner".
- `messaging` Passed to `get_stamenmap`. Default is FALSE.
- `...` Additional arguments passed on to `threshold` and `get_stamenmap`.

**See Also**

`get_stamenmap`, `ggmap`

**Examples**

```r
# Not run:
data('runs', package = 'trackeR')
plot_route(runs, session = 4, zoom = 13)
plot_route(runs, session = 4, zoom = 13, maptype = "terrain")
# multiple sessions
```
prepare_route

plot_route(runs, session = c(1:4, 8:11))
## different zoom level per panel
plot_route(runs, session = 6:7, zoom = c(13, 14))

## End(Not run)

---

**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**

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

**Arguments**

- `x` a `trackRdata` 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**

`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"))
```

print.trackerdata print method for trackerdata objects

Description

`print` method for `trackerdata` objects

Usage

```r
## S3 method for class 'trackerdata'
print(x, duration_unit = "h", digits = 2, ...)
```

Arguments

- `x` An object of class `trackerdata`.
- `duration_unit` The unit of duration in the resulting output. Default is h (hours).
- `digits` Number of digits to be printed.
- `...` Currently not used; only for compatibility with generic `summary` method only.

Details

The print method returns training coverage, number of sessions and total training duration from the data in the `trackerdata` object.
print.trackerdataSummary

Print method for session summaries.

Description

Print method for session summaries.

Usage

```r
## S3 method for class 'trackerdataSummary'
print(x, ..., digits = 2)
```

Arguments

- `x`: An object of class `trackerdataSummary`.
- `...`: Not used, for compatibility with generic summary method only.
- `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

```r
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 = 'trackerR')
dp <- distributionProfile(runs, what = 'speed')
dpFun <- profile2fd(dp, what = 'speed',
                    fdnames = list('speed', 'sessions', 'time above threshold'))
dp.pca <- pca.fd(dpFun, nhar = 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**

```r
readTCX(file, timezone = '', speedunit = "m_per_s",
         distanceunit = "m", ...)

readGXPX(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 a tcx, gpx, json or db3 file. Compressed versions (gz, bz2, xz, zip) of tcx, gpx, and json files are directly supported.
- **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.
**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, smooth_elevation_gain = TRUE,
      country = NULL, mask = TRUE, from_distances = NULL,
      correct_distances = FALSE, smooth_elevation_gain = TRUE,
      country = NULL, mask = TRUE, from_distances = NULL,
      correct_distances = FALSE, smooth_elevation_gain = TRUE,
      country = NULL, mask = TRUE, from_distances = NULL,
      correct_distances = FALSE, smooth_elevation_gain = TRUE,
      country = NULL, mask = TRUE, from_distances = NULL)
```
speedunit = NULL, distanceunit = NULL, sport = NULL, lgap = 30,
lskip = 5, m = 11, silent = FALSE)

readContainer(file, type = c("tcx", "gpx", "db3", "json"),
table = "gps_data", timezone = "", session_threshold = 2,
correct_distances = FALSE, smooth_elevation_gain = TRUE,
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 a tcx, gpx, json or db3 file. Compressed versions (gz, bz2, xz, zip) of tcx, gpx, and json files are directly supported.
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? Default is FALSE.
smooth_elevation_gain Logical. Should the elevation gain be smoothed before computing elevation gain? Default is TRUE.
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.
### 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`.

### Value

An object of class `trackerdata`.

### See Also

`trackerdata`, `readTCX`, `readDB3`, `readJSON`

### Examples

```r
filepath <- system.file("extdata/tcx", "2013-06-08-090442.TCX.gz", package = "trackR")
run <- read_container(filepath, type = "tcx", timezone = "GMT")
```
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.
aggregate 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.
smooth_elevation_gain Logical. Should the elevation gain be smoothed before computing elevation gain? Default is TRUE.
correct_distances Logical. Should the distances be corrected for elevation? Default is FALSE.
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.
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?

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 `trackRdata`.

See Also

`trackRdata`, `readTCX`, `readDB3`, `readJSON`

Examples

```r
## Not run:
filepath <- system.file("extdata/gpx", package = "trackR")
gpx_files <- read_directory(filepath)

## End(Not run)
```

ridges Generic function for ridgeline plots

Description

Generic function for ridgeline plots

Usage

```r
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

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, ...)

Examples
## Not run:

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

## End(Not run)
Arguments

x A `trackRdata` 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 = 'trackR')
ridges(runs)
## End(Not run)
```

run Training session.

Description

Training session.

Usage

run

Format

A `trackRdata` object containing one running training session.

runs Training sessions.

Description

Training sessions.

Usage

runs

Format

A `trackRdata` 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, ...)
```

## S3 method for class 'trackerData'
```r
call(session_duration(object, session = NULL, duration_unit = "h", ...))
```

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

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

Details
The times units will be inherited from object.

Session_times
Generic function for calculating session times

Description
Generic function for calculating session times

Usage
session_times(object, session, duration_unit, ...)

## S3 method for class 'trackRdata'
session_times(object, session = NULL, ...)

## S3 method for class 'trackRdataSummary'
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
smoother(object, ...)

Arguments
- object: The object to be smoothed.
- ...: Arguments to be passed to methods.
smoother.conProfile

Smoothing 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

Smoothing 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

Description

Smother 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)
## End(Not run)
```

`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)
```

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.Rdata

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

Description

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

Usage

smoother_control.Rdata(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.Rdata

sort.Rdata

Sort sessions in trackeRdata objects

Description

Sort the sessions trackeRdata objects into ascending or descending order according to the first session timestamp.
Usage

```r
## 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.

---

**speed2distance**  
*Convert speed to distance.*

Description

Convert speed to distance.

Usage

```r
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

```r
## S3 method for class 'trackerdata'
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`).
- `...`: Currently not used.

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'),
threshold.trackeRdata

Description

Thresholding for variables in trackeRdata objects

Usage

## S3 method for class 'trackeRdata'
threshold(object, variable, lower, upper, sport, 
trace = FALSE, ...)

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
- **trace** Should a progress report be printed? Default is FALSE
- ... 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
timeAboveThreshold

- heart rate [0, 250] bpm
- power [0, Inf] W
- pace [0, Inf] min per km
- duration [0, Inf] seconds
- temperature [-20, 60] C

after they have been transformed 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 = 'trackerR')
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**

```r
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)?
**timeline**  
*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 `trackeRdata` objects.

Timeline plot for `trackeRdataSummary` objects

**Usage**

```r
timeline(object, lims, ...)  
```

## S3 method for class 'trackeRdata'

timeline(object, lims = NULL, ...)

## S3 method for class 'trackeRdataSummary'

timeline(object, lims = NULL, ...)

**Arguments**

- **object**  
  An object of class `trackeRdata` or `trackeRdataSummary`.

- **lims**  
  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.

- **...**  
  Arguments passed to `summary.trackeRdata`.

**Examples**

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

## the same timeline plot applied on the `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 concentrationProfile and distributionProfile) 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.
Usage

`trackeRdata(dat, units = NULL, sport = NULL, session_threshold = 2,
           correct_distances = FALSE, smooth_elevation_gain = TRUE,
           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? Default is FALSE.
smooth_elevation_gain Logical. Should the elevation gain be smoothed before computing elevation gain? Default is TRUE.
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.

More details about the resulting `trackeRdata` object are available in the package vignette, which is an up-to-date version of Frick & Kosmidis (2017).
References


See Also

`readContainer` for reading .tcx and .db3 files directly into `trackeRdata` objects, and `get_elevation_gain` for details on the computation of the elevation gain.

Examples

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

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

### unique.trackeRdata

**Extract unique sessions in a trackeRdata object**

**Description**

Extract unique sessions in a trackeRdata object

**Usage**

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

**Arguments**

- `x` A trackeRdata object.
- `incomparables` Currently not used.
- `...` Currently not used.

**Details**

Uniqueness is determined by comparing the first timestamp of the sessions in the trackeRdata object.
**Wexp**

\[ W' \] expended.

**Description**

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

**Usage**

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

**Arguments**

- **object**
  Univariate `zoo` object containing the time stamped power output or speed values. (Power should be in Watts, speed in meters per second.)

- **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.

**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.
W'prime

References


W'prime

W': work capacity above critical power/speed.

Description

W': work capacity above critical power/speed.

Usage

W'prime(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:

$$W_{exp}(t_i) = W_{exp}(t_{i-1}) * exp(nu * (t_i - t_{i-1}))$$

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

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

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 `trackerWprime`.

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"), breaks = NULL,
     parallel = FALSE, n_zones = 9, unit_reference_sport = NULL, ...)

Arguments

object      An object of class trackerdata.
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).

Value
An object of class trackerdataZones.

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, 56
  - runs, 56
  - 1965, (Wprime), 73
  - 2012, (Wprime), 73
  - above (Wprime), 73
  - again, (Wprime), 73
  - al., (Wprime), 73
  - and (Wprime), 73
  - append, 4
  - append.trackedRdata, 4
  - applied (Wprime), 73
  - as.POSIXct, 48, 50, 52
  - available, (Wprime), 73
  - balance. (Wprime), 73
  - Based (Wprime), 73
  - been (Wprime), 73
  - below (Wprime), 73
  - bpm2bpm (conversions), 13
  - by (Wprime), 73
  - C2C (conversions), 13
  - c2d, 5
  - C2F (conversions), 13
  - capacity (Wprime), 73
  - change_units, 5
  - change_units.conProfile, 6
  - change_units.distrProfile, 6
  - change_units.trackedRdata, 7
  - change_units.trackedRdataSummary, 7
  - change_units.trackedRdataZones, 8
  - change_units.trackedRthresholds, 8
  - change_units.trackedRWprime, 9
  - changeUnits (change_units), 5
  - collect_units, 9
  - compute_breaks, 10
  - compute_limits, 10, 10
  - concentration_profile, 11, 30, 38, 54, 60
  - concentration_profile.distrProfile, 12
  - concentration_profile.trackedRdata
    - (concentration_profile.distrProfile), 12
  - concentrationProfile, 6, 28, 47, 69
  - concentrationProfile
    - (concentration_profile), 11
  - conProfile, 6, 21
  - conProfile
    - (concentration_profile.distrProfile), 12
  - conversions, 13
  - critical (Wprime), 73
  - cycling (Wprime), 73
  - data.frame, 45, 46, 70
  - Data2fd, 47
  - decreasing_smother, 18
  - decreasingSmoother
    - (decreasing_smother), 18
  - degree2degree (conversions), 13
  - depleted (Wprime), 73
  - describes (Wprime), 73
  - distance2speed, 19
  - distribution_profile, 19, 30, 38, 55, 61
  - distributionProfile, 6, 29, 47, 58, 69
  - distributionProfile
    - (distribution_profile), 19
  - distrProfile, 11, 12
  - distrProfile (distribution_profile), 19
  - during (Wprime), 73
  - et (Wprime), 73
  - exercise (Wprime), 73
  - expended, (Wprime), 73
  - F2C (conversions), 13
  - F2F (conversions), 13
  - fd, 47
  - find_unit_reference_sport, 21
finite (Wprime), 73
for (Wprime), 73
foreach, 53, 61, 73, 75
fortify_conProfile, 21
fortify_distrProfile, 22
fortify_trackerRdata, 22
fortify_trackerRdataSummary, 23
fortify_trackerWprime, 23
fortify_conProfile
  (fortify_conProfile), 21
fortify_distrProfile
  (fortify_distrProfile), 22
fortify_trackerRdata
  (fortify_trackerRdata), 22
fortify_trackerRdataSummary
  (fortify_trackerRdataSummary), 23
fortify_trackerWprime
  (fortify_trackerWprime), 23
ft2ft (conversions), 13
ft2km (conversions), 13
ft2m (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, 24, 42
GC2trackerRdata, 25
generate_thresholds, 26
generate_units, 9, 26, 46, 70
generateBaseUnits (generate_units), 26
generateDefaultThresholds
  (generate_thresholds), 26
get_elevation_gain, 27, 71
get_map, 44
get_operations, 28
get_operations.conProfile, 28
get_operations.distrProfile, 13, 20, 29
get_profile (get_profileRdata, 29
generate_thresholds, 25, 30
get_profile.distrProfile, 30
get_resting_periods, 30
generateThresholds, 25
get_sport, 21
get_sport (get_sport.trackeRWprime), 31
generate_thresholds (generate_thresholds), 26
get_operations (get_operations), 28
getUnits (get_units), 32
ggmap, 44
h2h (conversions), 13
h2min (conversions), 13
h2s (conversions), 13
h_per_km2min_per_km (conversions), 13
h_per_km2min_per_m (conversions), 13
h_per_m2min_per_km (conversions), 13
h_per_m2min_per_m (conversions), 13
has (Wprime), 73
how (Wprime), 73
impute_speeds, 35
impute_speeds (impute_speeds), 35
interest (Wprime), 73
is (Wprime), 73
it (Wprime), 73
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_h2m_per_s (conversions), 13
km_per_h2mi_per_h (conversions), 13
km_per_h2mi_per_min (conversions), 13
km_per_m2min_per_km (conversions), 13
INDEX

readJSON, 48
readTCX, 51, 53
readTCX (readX), 48
readX, 48, 50, 69
replenished (Wprime), 73
replenished (Wprime), 73
respectively (Wprime), 73
restingPeriods (get_resting_periods), 30
rev_per_min2rev_per_min (conversions), 13
rev_per_min2steps_per_min
  (conversions), 13
ridges, 53
ridges.conProfile, 54
ridges.distrProfile, 55
ridges.trackeRdata, 55
run, 56
runners (Wprime), 73
runs, 56
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, 57
scaled, 57
scaled.distrProfile, 58
scam, 18, 62
Scherrer, (Wprime), 73
session_duration, 58
session_times, 59
smoother, 59
smoother.conProfile, 60
smoother.distrProfile, 60, 62
smoother.trackeRdata, 61, 63
smoother_control.distrProfile, 38, 54, 55, 60, 61, 62
smoother_control.trackeRdata, 61, 62, 63
smootherControl.distrProfile
  (smoother_control.distrProfile), 62
smootherControl.trackeRdata, 39
smootherControl.trackeRdata
  (smoother_control.trackeRdata), 63
sort.trackeRdata, 63
speed (Wprime), 73
speed, (Wprime), 73
speed2distance, 64
steps_per_min2rev_per_min
  (conversions), 13
steps_per_min2steps_per_min
  (conversions), 13
still (Wprime), 73
substituting (Wprime), 73
summary, 46
summary.trackeRdata, 41, 64, 68
the (Wprime), 73
This (Wprime), 73
this (Wprime), 73
threshold, 36, 39, 44, 45
threshold (threshold_trackeRdata), 66
threshold.trackeRdata, 66
Thus, (Wprime), 73
timeAboveThreshold, 67
timeline, 68
to (Wprime), 73
traceR, 69
traceR-package (traceR), 69
traceeRdataSummary, 40, 68
traceeRdataSummary
  (summary_trackeRdata), 64
traceRW (Wprime), 9
trackeRW (Wprime), 73
unique.trackeRdata, 71
W (Wprime), 73
W' (Wprime), 73
W2KW (conversions), 13
W2W (conversions), 13
Wexp, 72
While (Wprime), 73
with (Wprime), 73
work (Wprime), 73
Wprime, 23, 43, 73
yet (Wprime), 73
zones, 41, 75
zoo, 35, 36, 72