trackeR: Infrastructure for Running, Cycling and Swimming Data from GPS-Enabled Tracking Devices in R

Hannah Frick
University College London

Ioannis Kosmidis
University of Warwick
The Alan Turing Institute

Abstract

This introduction to the R package **trackeR** is a modified version of Frick and Kosmidis (2017), published in the *Journal of Statistical Software*. The majority of changes and additions in the text are due to work by Ioannis Kosmidis, which involved the complete rewrite of most of the **trackeR** codebase at version 1.1.0, in order to make the package faster across the board and include support of all running, cycling and swimming and sport-specific units and variables, fix bugs, and implement new features and enhancements.

The use of GPS-enabled tracking devices and heart rate monitors is becoming increasingly common in sports and fitness activities. The **trackeR** package aims to fill the gap between the routine collection of data from such devices and their analyses in R. The package provides methods to import tracking data into data structures which preserve units of measurement and are organised in sessions. The package implements core infrastructure for relevant summaries and visualisations, as well as support for handling units of measurement. There are also methods for relevant analytic tools such as time spent in zones, work capacity above critical power (known as W'), and distribution and concentration profiles. A case study illustrates how the latter can be used to summarise the information from training sessions and use it in more advanced statistical analyses.

Keywords: sports, tracking, work capacity, running, cycling, distribution profiles.

1. Introduction

Recent technological advances allow the collection of detailed data on fitness activities and on multiple aspects of training and competition in professional sport. The focus of this paper is on data collected by GPS-enabled tracking devices and heart rate monitors. Such devices are routinely used in fitness activities such as running, cycling, and swimming and also during training in sports like field hockey and football. Basic questions associated with tracking data include how often, much, or hard an individual or a group trains, and a more advanced outlook tries to explain the impact of training on athlete physiology or performance.

Tools for basic analytics are usually offered by the manufacturers of the tracking devices, such as Garmin, Polar, and Catapult, and through a wide range of applications for devices such as smartphones and smartwatches, e.g., Strava Running and Cycling GPS, Endomondo – Running & Walking, and Runtastic Running GPS Tracker. A notable open-source effort is Golden Cheetah (`http://www.goldencheetah.org`), which has now, perhaps, become the gold standard in terms of facilities for importing tracking data from cycling activities and for associated analytics. However, Golden Cheetah is not designed to offer general flexibility in the statistical analysis of such sports tracking data.

The R system for statistical computing (R Core Team 2015) with its ecosystem of add-on packages provides a wide range of possibilities for the handling and analysis of tracking data.

GPS-enabled tracking devices typically record irregularly sampled spatio-temporal data. Infrastructure for such data is provided in the **trajectories** package (Pebesma and Klus 2015), which is developed around the "STIDF" class of the **spacetime** package (Pebesma 2012). However, the "STIDF" class does not accommodate missing values in positional or temporal information. Since this is commonly the case in data from GPS-enabled tracking devices (e.g., sequences of missing values in the positional data because the GPS signal is temporarily lost), a different approach is taken in **trackeR** (see Section 4). Other packages that offer tools for spatio-temporal data include **adehabtitatLT** (Calenge 2006), **trip** (Sumner 2015) and **move** (Kranstauber and Smolla 2015). The main focus of those packages is on animal tracking, e.g., estimation of habitat choices, and they are not directly suitable for tracking the various aspects of athlete activity.

Despite the wide range of R packages available, there is only a handful of packages specific to sport data and their analysis. The available packages focus on topics such as sports management (**RcmdrPlugin.SM**, Champely 2012), ranking sports teams (**mvglmmRank**, Karl and Broatch 2015), and accessing betting odds (**pinnacle.API**, Blume, Jhirad, and Gassem 2015). **SportsAnalytics** is a package that focuses on the analysis of performance data, and currently offers only "a selection of data sets, functions to fetch sports data, examples, and demos" (Eugster 2013). The **cycleRtools** package (Mackie 2015) provides functionality to import cycling data into R as well as tools for cycling-specific, descriptive analyses.

The **trackeR** package aims to fill the gap between the routine collection of data from GPS-enabled tracking devices and the analyses of such data within the R ecosystem. The package provides utilities to import sports data from GPS-enabled devices, and, after careful processing, organises them in data objects which are organised in separate sessions/workouts and carry information about the units of measurement (e.g., distance and speed units) as well as of any data operations that have been carried out (e.g., smoothing). The package also implements core infrastructure for the handling of measurement units and for summarising and visualising tracking data. It also provides functionality for calculating time in zones (e.g., Seiler and Kjerland 2006), work capacity W' (Skiba, Chidnok, Vanhatalo, and Jones 2012), and distribution and concentration profiles (Kosmidis and Passfield 2015), including a few methods for the analysis of these profiles.

Section 2 gives an overview of the package and introduces the basic objects and the methods that apply to them. Section 3 describes the importing utilities, and Section 4 details the structure and construction of the "trackeRdata" object, which is at the core of **trackeR**. Section 5 is devoted to the calculation of relevant summaries (time in zones, work capacity, distribution and concentration profiles) and the corresponding methods for visualisation. Section 6 and Section 7 focus on basic methods for unit manipulation as well as smoothing and thresholding. The case study in Section 8 investigates the key features in 27 sessions through a functional principal components analysis (e.g., Ramsay and Silverman 2005) on the concentration profiles for speed.

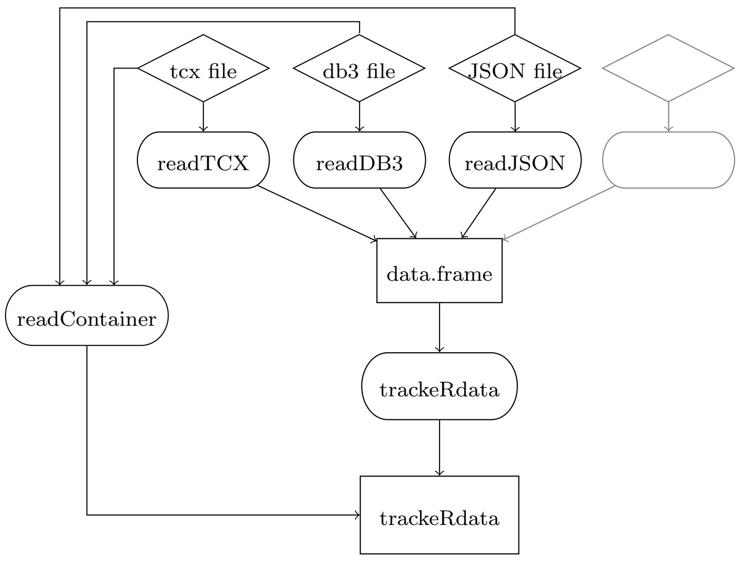

Figure 1: Package structure - Functionality to read tracking data.

2. Package structure

Figures 1 and 2 show a schematic overview of the package structure, split into two parts for reading data and further operations. Squared boxes indicate objects of a particular class, diamonds indicate files of a particular format, and boxes with rounded corners represent methods that apply to those objects. The respective class and method names are given in the boxes. An arrow from an object/file type to a method indicates that the method applies to objects of the respective class; an arrow from a method to an object indicates that the method outputs objects of the respective class. A bi-directional arrow between a method and an object indicates that the method's input and output are of the same class, such as the method `threshold()` and objects of class "`trackeRdata`". Arrows to or from groups of boxes apply to each box in the group. For example, the method `changeUnits()` applies to objects of classes "`trackeRdataZones`", "`trackeRdataSummary`", "`trackeRWprime`", "`distrProfile`", and "`conProfile`".

Data in various formats are imported and stored in the central data object of class "`trackeRdata`" from which summaries for descriptive purposes or further analyses can be derived. Methods for visualisation and data handling are available for data objects and summary objects. A list of all functionality is provided in Tables 1 and 2. For convenience, most camel-case function names in Tables 1 and 2 are aliased to function names were words are separated by underscores, e.g. `concentrationProfile()` and `concentration_profile()`, and `readContainer()` and `read_container()`.

3. Import utilities

trackeR provides utilities for data in common formats from GPS-enabled tracking devices. The family of the supplied reading functions, `read*()`, currently includes functions for reading TCX (Training Centre XML), GPX (as exported by Strava but other versions should work, too), DB3 (for SQLite, used, e.g., by devices from GPSports) and Golden Cheetah's

Function	Class	Description
`readTCX()`	TCX file	read TCX file
`readGPX()`	GPX file	read GPX file
`readDB3()`	DB3 file (SQLite)	read DB3 file
`readJSON()`	Golden Cheetah's JSON file	read JSON file
`readContainer()`	TCX/GPX/DB3/JSON file	read a TCX/GPX/DB3/JSON file
`readDirectory()`	TCX/GPX/DB3/JSON files	read all TCX/GPX/DB3/JSON files in a directory
`trackeRdata()`	"`data.frame`"	construct a "`trackeRdata`" object
`c()`	"`conProfile`", "`distrProfile`", "`trackeRdata`"	combine sessions
`sort()`	"`trackeRdata`"	sort sessions by start time
`unique()`	"`trackeRdata`"	extract unique sessions
`[]`	"`trackeRdata`"	subset sessions
`plot()`	"`trackeRdata`"	plot session profiles
`plotRoute()`	"`trackeRdata`"	plot route on a static map
`leafletRoute()`	"`trackeRdata`"	plot route on an interactive map
`threshold()`	"`trackeRdata`"	apply lower and upper bounds on data range
`smoother()`	"`conProfile`", "`distrProfile`", "`trackeRdata`"	smooth data by applying a summary function such as mean or median to a window
`getUnits()`	"`conProfile`", "`distrProfile`", "`trackeRdata`", "`trackeRdataSummary`", "`trackeRdataZones`", "`trackeRWprime`"	access units of measurement
`changeUnits()`	"`conProfile`", "`distrProfile`", "`trackeRdata`", "`trackeRdataSummary`", "`trackeRdataZones`", "`trackeRWprime`"	change units of measurement
`nsessions()`	"`conProfile`", "`distrProfile`", "`trackeRdata`", "`trackeRdataSummary`", "`trackeRdataZones`", "`trackeRWprime`"	number of sessions
`fortify()`	"`conProfile`", "`distrProfile`", "`trackeRdata`", "`trackeRdataSummary`", "`trackeRWprime`"	convert object into a data frame for plotting

Table 1: Functions available in the **trackeR** package (part 1).

Function	Class	Description
`summary()`	"`trackeRdata`"	summarise sessions
`print()`	"`trackeRdata`", "`trackeRdataSummary`"	print sessions summaries
`plot()`	"`conProfile`", "`distrProfile`", "`trackeRdata`", "`trackeRdataSummary`", "`trackeRdataZones`", "`trackeRfpca`", "`trackeRWprime`"	
`timeline()`	"`trackeRdata`", "`trackeRdataSummary`"	plot timeline summary
`zones()`	"`trackeRdata`"	time spent in zones
`Wprime()`	"`trackeRdata`"	calculate W' *balance* or W' *expended*
`plot()`	"`trackeRWprime`"	plot W' *balance* or W' *expended*
`session_times()`	"`trackeRdata`", "`trackeRdataSummary`"	return the start and end date and time of its session
`session_duration()`	"`trackeRdata`", "`trackeRdataSummary`"	return the duration of each session
`get_sport()`	"`conProfile`", "`distrProfile`", "`trackeRdata`", "`trackeRdataSummary`", "`trackeRWprime`"	return the sports of each session in the object
`concentrationProfile()`	"`distrProfile`", "`trackeRdata`"	calculate concentration profiles
`distributionProfile()`	"`trackeRdata`"	calculate distribution profiles
`ridges()`	"`conProfile`", "`distrProfile`", "`trackeRdata`"	ridgeline plots of concentration/distribution profiles
`profile2fd()`	"`conProfile`", "`distrProfile`"	convert profiles to "`fd`" class
`funPCA()`	"`conProfile`", "`distrProfile`"	functional principal components analysis

Table 2: Functions available in the **trackeR** package (part 2).

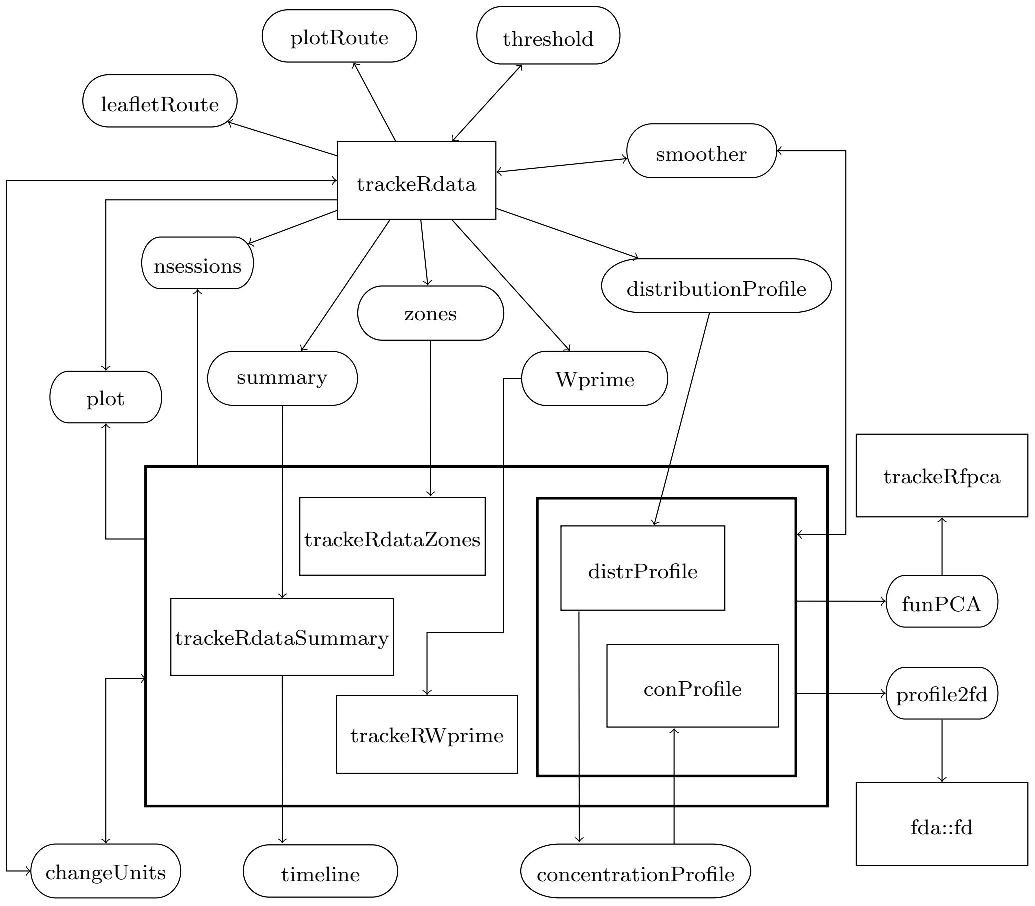

Figure 2: Package structure - Functionality to analyse tracking data.

JSON files. These functions read the tracking data, and return a `data.frame` with a specific structure.

The following code chunk illustrates the use of the `readTCX()` function using a TCX file that ships with the package and shows the name and type of variables that are present in the resulting data frame.

```
R> filepath <- system.file("extdata/tcx", "2013-06-01-183220.TCX.gz",
+                            package = "trackeR")
R> runDF <- readTCX(file = filepath, timezone = "GMT")
R> str(runDF)
```

```
'data.frame':        3881 obs. of  11 variables:
 $ time             : POSIXct, format: "2013-06-01 17:32:20" ...
 $ latitude         : num  50.8 50.8 50.8 50.8 50.8 ...
 $ longitude        : num  -1.7 -1.7 -1.7 -1.7 -1.7 ...
 $ altitude         : num  83.4 83.8 84 83.8 83.6 ...
 $ distance         : num  1.26 3.3 7.12 11.12 16.76 ...
 $ heart_rate       : num  56 61 61 71 71 74 74 85 85 85 ...
 $ speed            : num  0.885 1.209 1.801 2.205 2.756 ...
 $ cadence_running: num  60 63 70 78 83 84 84 85 85 86 ...
 $ cadence_cycling: logi  NA NA NA NA NA NA ...
 $ power          : logi  NA NA NA NA NA NA ...
 $ temperature    : logi  NA NA NA NA NA NA ...
 - attr(*, "sport")= chr "running"
 - attr(*, "file")= chr "/private/var/folders/3t/00tlvfn14zq5v45q3q3y63cm0000gn/T/RtmpZ5Y4
```

Power is not available in the above data frame because the data has been identified to come from a running training by regex matching its contents with a sport-determining list of keywords. Times are taken here to be in GMT. The default for argument `timezone` is `""` and is system-specific; see `?as.POSIXct` for details.

trackeR can accommodate the addition of extra formats by simply authoring appropriate import functions. Such functions should take as input the path of the file to be read and return a data frame with the same structure and attributes as in the above example.

4. "`trackeRdata`" class

4.1. Object structure

The core object of **trackeR** has class "`trackeRdata`". The "`trackeRdata`" objects are session-based, unit-aware and operation-aware structures, which organise the data in a list of multivariate `zoo` objects (Zeileis and Grothendieck 2005) with one element per session. The observations within each session are ordered according to the time stamps as these are read from the GPS-enabled tracking devices. Each "`trackeRdata`" object has an attribute on the measurement units of the data it holds, and, if applicable, an attribute detailing the operations, such as smoothing, it has gone through.

"`trackeRdata`" objects result from the constructor function `trackeRdata()`, which takes as input the output of the `read*()` functions. Apart from the allocation of observations into distinct sessions, the constructor function also performs some data processing, including basic sanity checks (for example, removing observations with negative or missing values for cumulative distance or speed), handling of measurement units, correction of distances using altitude data if required, and data imputation, discussed in Section 4.5.

4.2. Constructor function

The interface of the constructor function for class "`trackeRdata`" is

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

`dat` is the data frame containing the tracking data and `units` is used to specify the units of measurement. Table 3 shows the currently supported units and notes the units that are used by default when `units = NULL`. The argument `sport` indicate the sport from which the data is coming from and must be "running", "cycling", "swimming". This affects the calculation of W' (based on power or speed for cycling and running, respectively) and the thresholds applied before plotting the session data. The other arguments are specific to the data processing operations, which are briefly described in the following subsections.

4.3. Identifying distinct sessions

The constructor function groups the observations into sessions according to their time stamps. Specifically, the time stamps in the data from the `read*()` functions are first sorted, and all consecutive observations whose time stamps are no further apart from each other than a specified threshold t^* are considered to belong to a distinct session. The value of t^* is set via the `session_threshold` argument of the `trackeRdata()` function and it defaults to 2 hours.

4.4. Distance correction using altitude data

If the distances in the data have been calculated solely based on latitude and longitude data, without taking into account the altitude, then the distance covered can be underestimated. The `correct_distances` argument of the `trackeRdata()` function controls whether the distances should be corrected for altitude changes.

If the uncorrected distance covered at time point t_i is $d_{2,i}$, then setting `correct_distances = TRUE` uses the Pythagorean theorem to correct the distance covered between time point t_{i-1} and time point t_i to

$$d_i - d_{i-1} = \sqrt{(d_{2,i} - d_{2,i-1})^2 + (a_i - a_{i-1})^2}\,,$$

where d_i and a_i are the corrected cumulative distance and the altitude at time t_i, respectively.

If no altitude measurements are available, these are extracted from *SRTM 90m Digital Elevation Data* via the **raster** package (Hijmans 2015) using the latitude and longitude measurements. The arguments `country` and `mask` control the extraction of altitudes.

4.5. Imputation process

Occasionally, there is a large time difference between consecutive observations in the same session, sometimes of the order of several minutes. This can happen, for example, if the device is intentionally paused by the athlete or if the proprietary algorithm controlling the operating sampling rate of the device detects no significant change in position. For example, in the manual of a GPS device, the Forerunner© 310XT, it is stated that *"The Forerunner uses smart recording. It records key points when you change direction, speed, or heart rate"* (Garmin Ltd. 2013). In both cases, interpolating directly to get the speed or power will lead to overestimation of the total workload within those intervals.

We assume that such intervals appear only when there is no significant work happening, and hence impute them with observations with zero speed (for running) or zero speed and power (for cycling).

Figure 3 shows a schematic representation of the imputation process for speed. The parameters l_{gap}, m and l_{skip} control the imputation, and can be specified via the `lgap`, `m` and `lskip` arguments of the `trackeRdata()` function, respectively.

If the observations at times t_i and t_{i+1} are more than l_{gap} seconds apart, then it is assumed that there is no significant work happening between t_i and t_{i+1}. The number of imputed records in the interval is m, and consists of two 'outer' records and $m-2$ 'inner' records. The

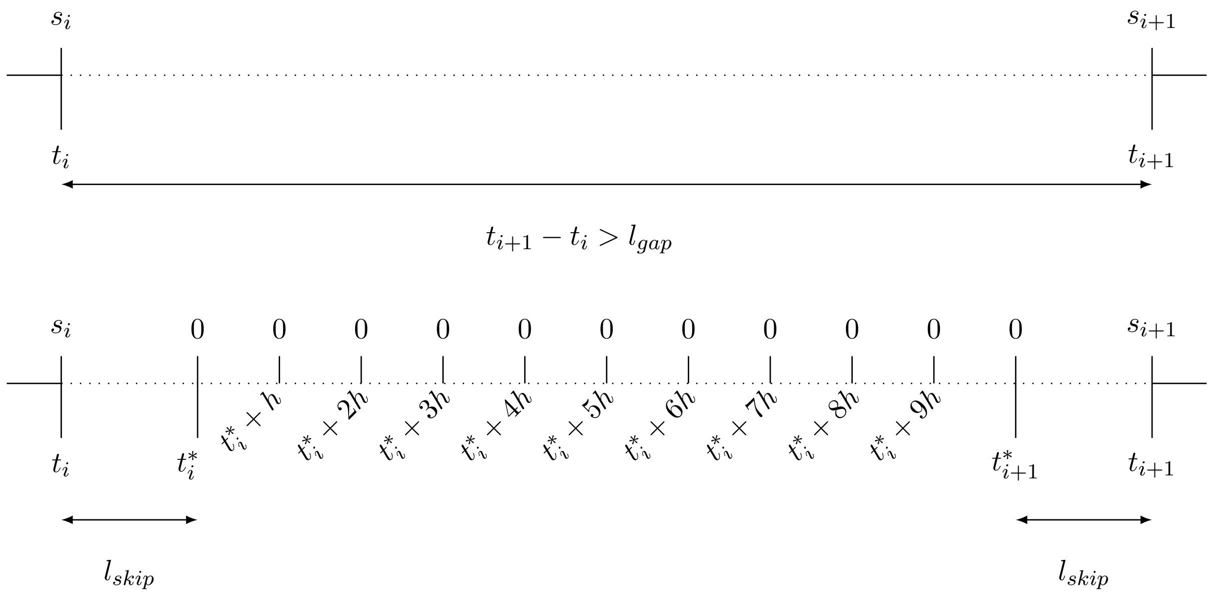

Figure 3: Illustration of the imputation process for speed with $m = 11$.

'outer' records are l_{skip} seconds apart from the existing observations forming the beginning and the end of the interval, respectively. The 'inner' records are $h = (t_{i+1} - t_i - 2l_{skip})/(m-1)$ seconds apart.

The imputed records between t_i and t_{i+1} have zero speed or power, and the latitude, longitude and altitude measurements are set to their values at time t_i. All other variables are set to `NA`.

`trackeRdata()` also adds five records at the beginning and five at the end of a session, based on the assumption that there is no activity before and after the available records. These observations have zero speed or power, their latitude, longitude and altitude measurements are as in the first and last observations, respectively, and all other variables are set to `NA`. The imputed records are one second apart from each other and from the first and the last observation, respectively.

After the imputation process, the cumulative distances are updated based on the imputed speeds and the time differences between consecutive observations, according to

$$d_{i+1} = d_i + s_i(t_{i+1} - t_i)$$

where s_i and d_i denote the speed and cumulative distance at time point t_i, respectively.

The following code chunk takes as input the raw data in the data frame `runDF` and constructs the corresponding "`trackeRdata`" object.

```
R> runTr0 <- trackeRdata(runDF)
```

The `print()` method for "`trackeRdata`" objects displays some basic summaries about the sessions, including the sports present in the sessions, the number of sessions, training coverage, total training duration and the units for each of the variables.

```
R> runTr0
```

```
A trackeRdata object
Sports: running

Training coverage: from 2013-06-01 18:32:15 to 2013-06-01 19:37:56
Number of sessions: 1
Training duration: 1.09 h

Units

 latitude        degree        cycling
 longitude       degree        cycling
 altitude        m             cycling
 distance        m             cycling
 heart_rate      bpm           cycling
 speed           m_per_s       cycling
 cadence_cycling rev_per_min   cycling
 power           W             cycling
 temperature     C             cycling
 pace            min_per_km    cycling
 duration        min           cycling
 latitude        degree        running
 longitude       degree        running
 altitude        m             running
 distance        m             running
 heart_rate      bpm           running
 speed           m_per_s       running
 cadence_running steps_per_min running
 temperature     C             running
 pace            min_per_km    running
 duration        min           running
 latitude        degree        swimming
 longitude       degree        swimming
 altitude        m             swimming
 distance        m             swimming
 heart_rate      bpm           swimming
 speed           m_per_s       swimming
 temperature     C             swimming
 pace            min_per_km    swimming
 duration        min           swimming
```

The function `readContainer()` is a convenience wrapper that calls the suitable reading function and, then, `trackeRdata()` for the data processing and the organisation of the data in a "`trackeRdata`" object (see `?readContainer` for the available arguments). For example,

```
R> runTr1 <- readContainer(filepath, type = "tcx", timezone = "GMT")
R> identical(runTr0, runTr1)

[1] TRUE
```

The function `readDirectory()` allows the user to read all files of a supported format in a directory, rather than calling, e.g., `readContainer()` on each file separately. For example, the following chunk of code will read a directory with one cycling, one running and, one swimming session, constructs the corresponding trackeRdata object, and then extracts the sports for each session.

```
R> gpxDir <- system.file("extdata/gpx", package = "trackeR")
R> workouts <- readDirectory(gpxDir, verbose = FALSE)
R> get_sport(workouts)
```

```
[1] "running"  "cycling"  "swimming"
```

Using the argument `aggregate`, the user can decide if all data are first combined in a data frame and then split into sessions solely based on the time difference between consecutive observations. This way, e.g., warm-up and cool-down phases are put into the same session as the central part of training, even if they are recorded in separate container files. Alternatively, data from different container files are always stored in separate sessions.

trackeR ships with two "`trackeRdata`" objects containing 1 and 27 running sessions, respectively, and which can be loaded via

```
R> data("run", package = "trackeR")
R> data("runs", package = "trackeR")
```

We will use those objects for the illustrations throughout the paper.

5. Session summaries and visualisation

trackeR provides methods for summarising sessions in terms of scalar summaries, the time spent exercising in specified zones, the concept of work capacity, and distribution and concentration profiles.

5.1. Visualisation

For a first visual inspection of the data, the `plot()` method shows by default the evolution of heart rate and pace over the course of the selected sessions. For example, Figure 4 shows the evolution of heart rate and pace for the first three sessions in the `runs` object.

```
R> plot(runs, session = 1:3)
```

The route covered during a session can also be displayed on a static map via the `plotRoute()` method. The `plotRoute()` method uses the **ggmap** package (Kahle and Wickham 2013) and, hence, can work with the sources and maps supported by **ggmap**. For example, Figure 5 shows the route covered during session 4 in `runs` using a map downloaded from Google. Interactive maps can be produced with `leafletPlot()`, using the **leaflet** package (Cheng and Xie 2016).

```
R> plotRoute(runs, session = 4, zoom = 13)
```

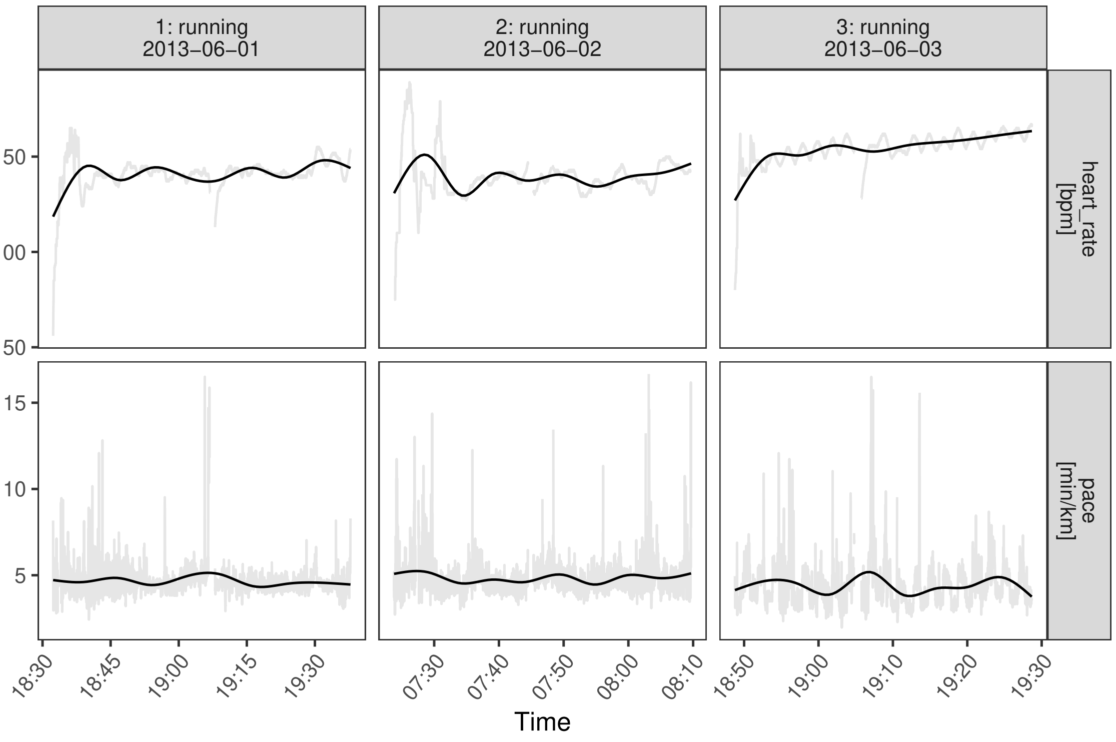

Figure 4: Heart rate and pace over the course of sessions 1–3.

5.2. Scalar summaries

Each session can be summarised through common summary statistics using the `summary()` method. Such a session summary includes estimates of the total distance covered, the total duration, the time spent moving, and work to rest ratio. It also includes averages of speed, pace, cadence, power, and heart rate, calculated based on total duration or the time spent moving.

An athlete is considered to be moving if the speed is larger than some threshold s^*. This threshold can be set via the `moving_threshold` argument of the `summary()` method, and the package assumes that anything between not moving at all and walking with a speed below that threshold is resting. The default value for `moving_threshold` has been set to 1 meter per second for running, which is just below the speed humans prefer to walk at on average (1.4 meters per second; see Bohannon 1997), 0.5 meter per second for swimming and 2 meters per second for cycling.

The "average speed moving" is calculated as total distance covered divided by time moving while "average speed" is calculated as total distance divided by total duration. The average pace (moving) is calculated as the inverse of the average speed (moving). The work to rest ratio is calculated as time moving divided by (total duration - time moving). The averages for cadence, power, and heart rate (total and moving) are weighted averages with weights depending on the time difference to the next observation. These averages also need to take into account missingness in the observations. For a variable of interest V, we can calculate a

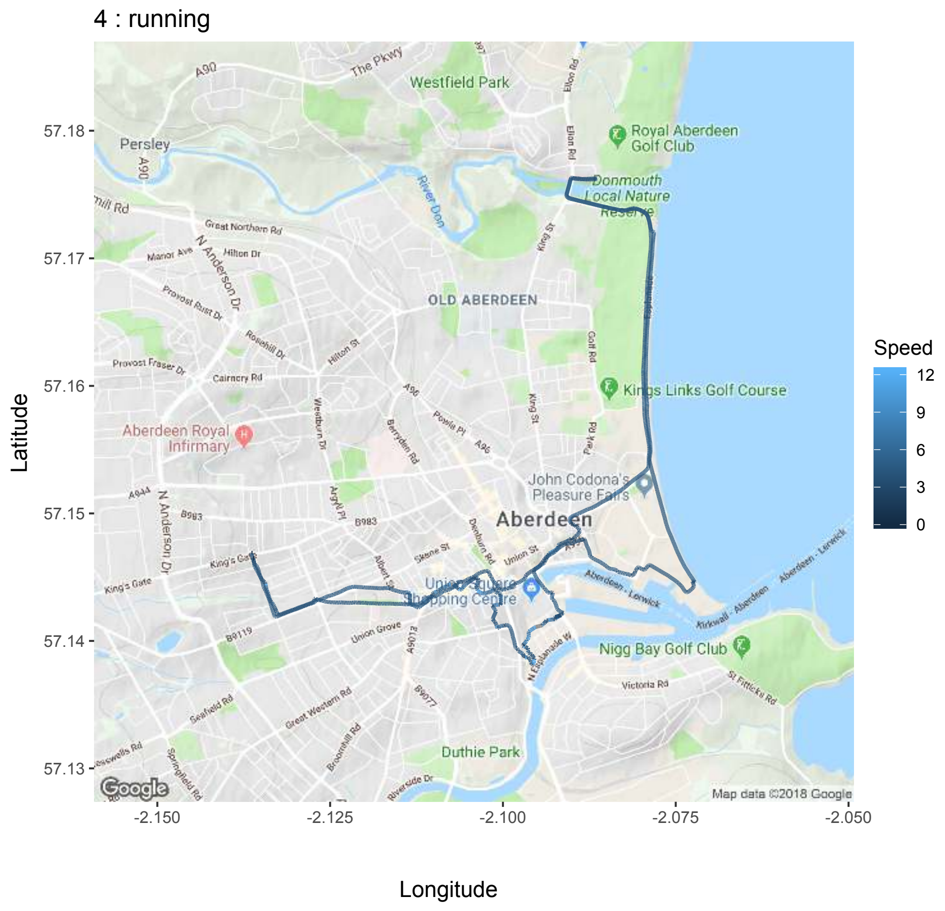

Figure 5: Route covered during session 4 on a map from Google.

weighted mean for the total session while accounting for missing values via

$$\sum_i v_i \frac{\Delta_i K_i}{\sum_i \Delta_i K_i}$$

and its counterpart for the part of the session spent in motion via

$$\sum_i v_i \frac{\Delta_i K_i I(s_i > s^*)}{\sum_i \Delta_i K_i I(s_i > s^*)}$$

where v_i is the value of V at time point t_i, K_i is 1 if v_i is available, i.e., not missing, and 0 otherwise, $I(\cdot)$ denotes the indicator function, and $\Delta_i = t_i - t_{i-1}$ the time difference between observations at t_i and t_{i-1}.

The `summary()` method for "`trackeRdata`" objects returns a data frame which can be used for further analysis. The return object is classed as "`trackeRdataSummary`" for which several methods are available. With the `print()` method, one can set the number of digits printed for the scalar summary statistics. The following example shows the summaries for sessions 1–2 with the default number of digits of 2 and then the summary of session 1 with 3 digits for comparison.

```
R> summary(runs, session = 1:2)

 *** Session 1 : running ***

 Session times: 2013-06-01 18:32:15 - 2013-06-01 19:37:56
 Distance: 14130.7 m
 Duration: 65.68 mins
 Moving time: 64.17 mins
 Average speed: 3.59 m_per_s
 Average speed moving: 3.67 m_per_s
 Average pace (per 1 km): 4:38 min:sec
 Average pace moving (per 1 km): 4:32 min:sec
 Average cadence running: 88.66 steps_per_min
 Average cadence cycling: NA rev_per_min
 Average cadence running moving: 88.87 steps_per_min
 Average cadence cycling moving: NA rev_per_min
 Average power: NA W
 Average power moving: NA W
 Average heart rate: 141.11 bpm
 Average heart rate moving: 141.13 bpm
 Average heart rate resting: 136.76 bpm
 Average temperature: NA C
 Total elevation gain: 94.2 m
 Work to rest ratio: 42.31

 *** Session 2 : running ***

 Session times: 2013-06-02 07:23:43 - 2013-06-02 08:09:47
 Distance: 9450.24 m
 Duration: 46.07 mins
 Moving time: 44.13 mins
 Average speed: 3.42 m_per_s
 Average speed moving: 3.57 m_per_s
 Average pace (per 1 km): 4:52 min:sec
 Average pace moving (per 1 km): 4:40 min:sec
 Average cadence running: 88.21 steps_per_min
 Average cadence cycling: NA rev_per_min
 Average cadence running moving: 88.25 steps_per_min
 Average cadence cycling moving: NA rev_per_min
 Average power: NA W
 Average power moving: NA W
 Average heart rate: 139.48 bpm
 Average heart rate moving: 139.44 bpm
 Average heart rate resting: 141.16 bpm
 Average temperature: NA C
 Total elevation gain: 124.52 m
 Work to rest ratio: 22.83
 Moving thresholds: 2.0 (cycling) 1.0 (running) 0.5 (swimming) m_per_s
 Unit reference sport: running
```

```
R> runSummary <- summary(runs, session = 1)
R> print(runSummary, digits = 3)

 *** Session 1 : running ***

 Session times: 2013-06-01 18:32:15 - 2013-06-01 19:37:56
 Distance: 14130.7 m
 Duration: 65.683 mins
 Moving time: 64.167 mins
 Average speed: 3.586 m_per_s
 Average speed moving: 3.67 m_per_s
 Average pace (per 1 km): 4:38 min:sec
 Average pace moving (per 1 km): 4:32 min:sec
 Average cadence running: 88.664 steps_per_min
 Average cadence cycling: NA rev_per_min
 Average cadence running moving: 88.874 steps_per_min
 Average cadence cycling moving: NA rev_per_min
 Average power: NA W
 Average power moving: NA W
 Average heart rate: 141.107 bpm
 Average heart rate moving: 141.131 bpm
 Average heart rate resting: 136.762 bpm
 Average temperature: NA C
 Total elevation gain: 94.196 m
 Work to rest ratio: 42.308

 Moving thresholds: 2.0 (cycling) 1.0 (running) 0.5 (swimming) m_per_s
 Unit reference sport: running
```

The `plot()` method shows the evolution of the various summary statistics over calender time or over the course of the sessions. For example, the following code chunk produces Figure 6.

```
R> runSummaryFull <- summary(runs)
R> plot(runSummaryFull, group = c("total", "moving"),
+    what = c("avgSpeed", "distance", "duration", "avgHeartRate"))
```

5.3. Times in zones

A common way to summarise and characterise a session is to calculate how much time was spent exercising in certain zones, e.g., heart rate zones.

The `zones()` method for sessions returns an object of class "`trackeRdataZones`" for which methods `changeUnits()` and `plot()` are provided. The user can specify the variables, such

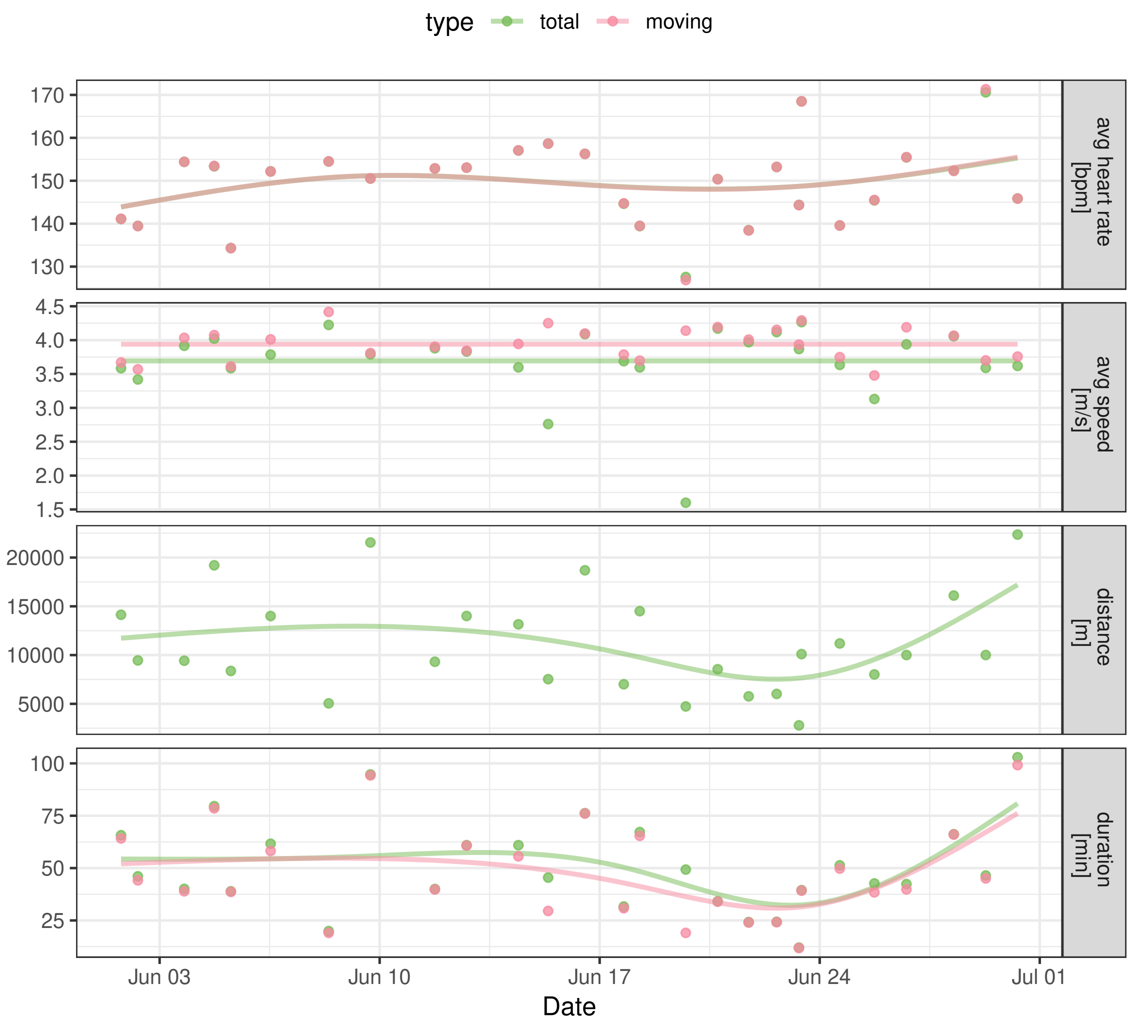

Figure 6: Selected session summaries for all 27 sessions.

as heart rate and speed, and their respective zones via the arguments `what` and `breaks`, respectively. Figure 7 shows a graphical representation of the zones summary, making it easier to see that more (relative) time was spent training with high speed ($> 4\,m/s$) in sessions 3 and 4 than in sessions 1 and 2. The following code chunk illustrates three equivalent ways to specify the zones for a single variable: 1) in the standard way through arguments `what` and `breaks` 2) if `breaks` is a named list, argument `what` can be left unspecified and 3) if only a single variable is to be evaluated, `breaks` can also be a vector.

```
R> runZones <- zones(runs[1:4], what = "speed",
+    breaks = list(speed = c(0, 2:6, 12.5)))
R> runZones <- zones(runs[1:4], breaks = list(speed = c(0, 2:6, 12.5)))
R> runZones <- zones(runs[1:4], what = "speed", breaks = c(0, 2:6, 12.5))
R> plot(runZones)
```

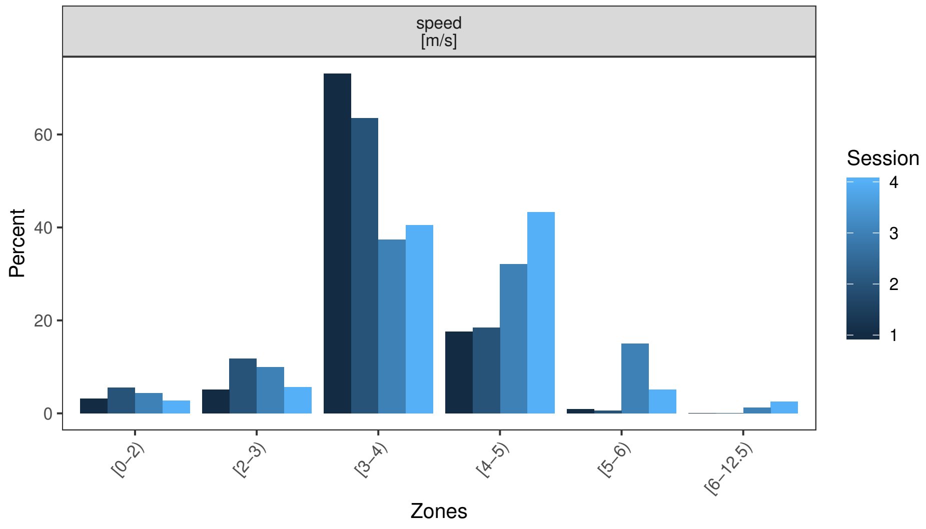

Figure 7: Zone summaries for speed of sessions 1–4.

5.4. Quantifying work capacity

The critical power model (Monod and Scherrer 1965) describes the relationship between the power output P and the time t_e to exhaustion at that power output

$$P = (W'_0/t_e) + CP \tag{1}$$

in terms of two parameters W'_0 and CP. The critical power (CP) is defined by Monod and Scherrer (1965) as “the maximum rate (of work) that [can be kept] up for a very long time without fatigue.” Skiba *et al.* (2012) describe CP as “a power output that could *theoretically* be maintained indefinitely on the basis of principally ’aerobic’ metabolism.” W' (read W prime) represents a finite work capacity above CP. Skiba *et al.* (2012) assume that W' gets depleted during exercise with a power output above CP but also replenished during exercise with a power output of or below CP. We denote as W' the general concept of work capacity above CP, and $W'(t)$ is the state of W' at time t. The latter is also sometimes referred to as W' *balance* at time t. Additionally, the initial state of W' at the start of an exercise $t = t_0$ is $W'_0 = W'(t_0)$, which is one of the parameters in the critical power model (Equation 1). Total depletion of W'_0 results in the inability to produce a power output above CP. Thus, knowledge of the current state $W'(t)$, i.e., how much of that finite work capacity W'_0 is left at time t, is important to an athlete, particularly in a race.

While this concept is most commonly applied to cycling, where the power output is routinely measured, Skiba *et al.* (2012) suggest that it can also be applied to running, substituting power and critical power by speed and critical speed, respectively. For running, the model postulates that each runner has a finite capacity in terms of distance covered above the critical speed. Depending on how much the runner exceeds this critical speed, the finite capacity W'_0 is being exhausted in shorter times. Below we describe the models for depletion and replenishment of work capacity and how they are combined in **trackeR**.

Depletion of work capacity

Assuming constant power for periods of exertion above CP, Skiba, Fulford, Clarke, Vanhatalo, and Jones (2015) assume that W' is depleted at a rate directly proportional to the difference between the power output and CP

$$\frac{d}{dt}W'(t) = -(P - CP)\,. \tag{2}$$

Solving Equation 2 for $W'(t)$ gives

$$W'(t) = -(P - CP)t + D \tag{3}$$

where $D \in \mathbb{R}$ is constant over t.

Suppose that the exercise over time $t_0 = 0$ to $t_n = T$ can be split into n intervals with breakpoints $t_0, t_1, \ldots, t_n$ such that the power output within each interval is constant, that is $P(t) = P_i$ for $t \in [t_{i-1}, t_i)$, $i \in \{1, \ldots, n\}$. Then, using Equation 3, the change in $W'(t)$ over the interval can be expressed as

$$W'(t_i) - W'(t_{i-1}) = -(P_i - CP)(t_i - t_{i-1})\,. \tag{4}$$

Replenishing of work capacity

Skiba *et al.* (2015) assume that the periods with a power output at or below CP are periods of recovery during which W' is replenished with a rate that depends on the difference between CP and the power output, and the amount of W'_0 remaining, as follows:

$$\frac{d}{dt}W'(t) = \left(1 - \frac{W'(t)}{W'_0}\right)(CP - P)\,. \tag{5}$$

Equation 5 assumes that recovery slows down as $W'(t)$ approaches the initial capacity W'_0. Employing the substitution rule for integrals while solving Equation 5 and reexpressing in terms of $W'(t_{i-1})$ (see Appendix A for details) gives

$$W'(t_i) = W'_0 - \left(W'_0 - W'(t_{i-1})\right)\exp\left(\frac{P_i - CP}{W'_0}(t_i - t_{i-1})\right)\,. \tag{6}$$

Since $W'(t_{i-1})$ is the amount of W'_0 remaining at the start of the interval $[t_{i-1}, t_i)$, $W'_0 - W'(t_{i-1})$ is the amount of W'_0 which has been depleted prior to t_{i-1} and not yet been replenished. Skiba *et al.* (2012) refer to this as W' *expended*. Skiba *et al.* (2015) describe the replenishing of W' indirectly by describing how W' *expended* is reduced over the course of such a recovery interval. The exponential decay factor used in Equation 6 here is the same as their Equation 4 with only different notation. Skiba *et al.* (2015) use t to describe the length of the interval, $D_{CP} = CP - P_i$ for the difference between critical power and power output, and W'_{exp} for the amount of W' previously expended. For $P_i < CP$, as is required for replenishment, $-D_{CP}$ and $P_i - CP$ are negative and thus the exponential factor is smaller than 1, leading to an exponential decay as described.

Skiba *et al.* (2012) also assume an exponential decay of previously expended W' to describe replenishing W', albeit with a different decay factor. Instead of $(P_i - CP)/W'_0$, they use $1/\tau_{W'}$. The relationship between the time constant of replenishing $\tau_{W'}$ and the difference between critical power and recovery power $\bar{P}$ is estimated based on experimental data as

$$\tau_{W'} = 546\exp\left(-0.01(CP - \bar{P})\right) + 316$$

with recovery power $\bar{P}$ estimated by the mean of all power outputs below CP.

Using Equation 6, i.e., the formulation of Skiba *et al.* (2015), the change in W' over the corresponding interval $[t_{i-1}, t_i)$ can be described through

$$W'(t_i) - W'(t_{i-1}) = (W'_0 - W'(t_{i-1})) \left(1 - \exp\left(\frac{P_i - CP}{W'_0}\Delta_i\right)\right) . \tag{7}$$

Work capacity at time t_j

Equation 4 describes the depletion of W' (when $P_i > CP$) and Equation 7 describes replenishment of W' (when $P_i \leq CP$) over an interval $[t_{i-1}, t_i)$. These two aspects can be combined to describe the change over the interval as

$$\begin{aligned} W'(t_i) - W'(t_{i-1}) = & -(P_i - CP)\Delta_i I(P_i > CP) + \\ & (W'_0 - W'(t_{i-1})) \left(1 - \exp\left(\frac{P_i - CP}{W'_0}\Delta_i\right)\right)(1 - I(P_i > CP)) . \end{aligned}$$

The amount of W' left at time point t_j, $j \in \{1, \ldots, n\}$, can thus be described through the initial amount W'_0 and the changes happening in the j intervals of constant power previous to t_j:

$$\begin{aligned} W'(t_j) &= W'_0 + \sum_{i=1}^{j} (W'(t_i) - W'(t_{i-1})) \\ &= W'_0 - \sum_{i=1}^{j} (P_i - CP)\Delta_i I(P_i > CP) + \\ & \quad \sum_{i=1}^{j} (W'_0 - W'(t_{i-1})) \left(1 - \exp\left(\frac{P_i - CP}{W'_0}\Delta_i\right)\right)(1 - I(P_i > CP)) . \end{aligned} \tag{8}$$

W' *expended* at time t_j is then $W'_0 - W'(t_j)$.

Function `Wprime()` can be used to calculate W' *expended* by setting argument `quantity` to `"expended"`. If `quantity` is set to `"balance"`, `Wprime()` calculates the current state $W'(t)$ (Equation 8). `Wprime()` contains implementations for Skiba *et al.* (2012) and Skiba *et al.* (2015), which can be selected via the `version` argument. For example, session 11 of the example data is an interval training with a warm-up and cool-down phase. Assuming a critical speed of 4 meters per second, the following code chunk produces Figure 8, which shows W' *expended*, based on the specification of Skiba *et al.* (2012), along with the corresponding speed profile.

```
R> wexp <- Wprime(runs, session = 11, quantity = "expended",
+    cp = 4, version = "2012")
R> plot(wexp, scaled = TRUE)
```

During the warm-up phase speed rarely exceeds 4 meters per second and W' *expended* remains low. Over the course of the interval training, W' *expended* rises during the high-intensity

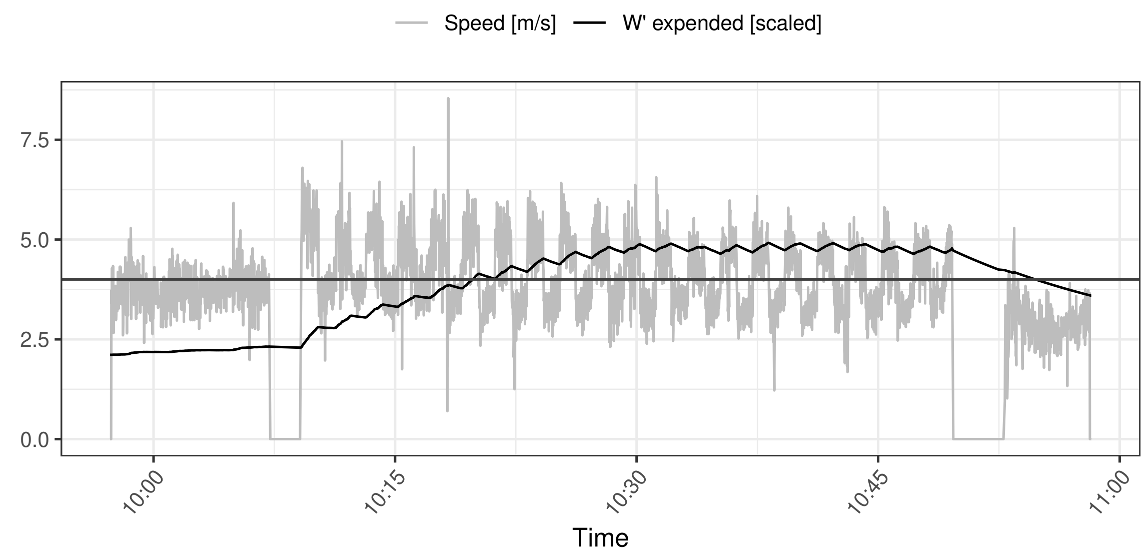

Figure 8: W' expended in session 11.

phases and drops during the recovery phases. In the last part of the session, speeds are mostly below 4 meters per second and W' *expended* drops again.

5.5. Distribution and concentration profiles

Kosmidis and Passfield (2015) introduce the concept of distribution profiles for which the **trackeR** package provides an implementation. These profiles are motivated by the need to compare sessions and use information on such variables as heart rate or speed during a session for further modelling.

For a session lasting t_n seconds, the distribution profile is defined as the curve $\{v, \Pi(v) | v \geq 0\}$ where

$$\Pi(v) = \int_0^{t_n} I(v(t) > v)dt\,.$$

The function $\Pi(v)$ is monotone decreasing and describes the time spent exercising above a threshold v for a variable V under consideration (e.g., heart rate or speed).

On the basis of observations $v_0, \ldots, v_n$ for V, at respective time points $t_0, \ldots, t_n$, the observed version of $\Pi(v)$ can be calculated as

$$P(v) = \sum_{i=1}^{n} (t_i - t_{i-1}) I(v_i > v)\,.$$

This can subsequently be smoothed respecting the positivity and monotonicity of $\Pi(v)$, e.g., via a shape constraint additive model with Poisson responses (Pya and Wood 2015).

The concentration profile is defined in Kosmidis and Passfield (2015) as the negative derivative of a distribution profile and is suitable for revealing concentrations of time around certain values of the variable under consideration.

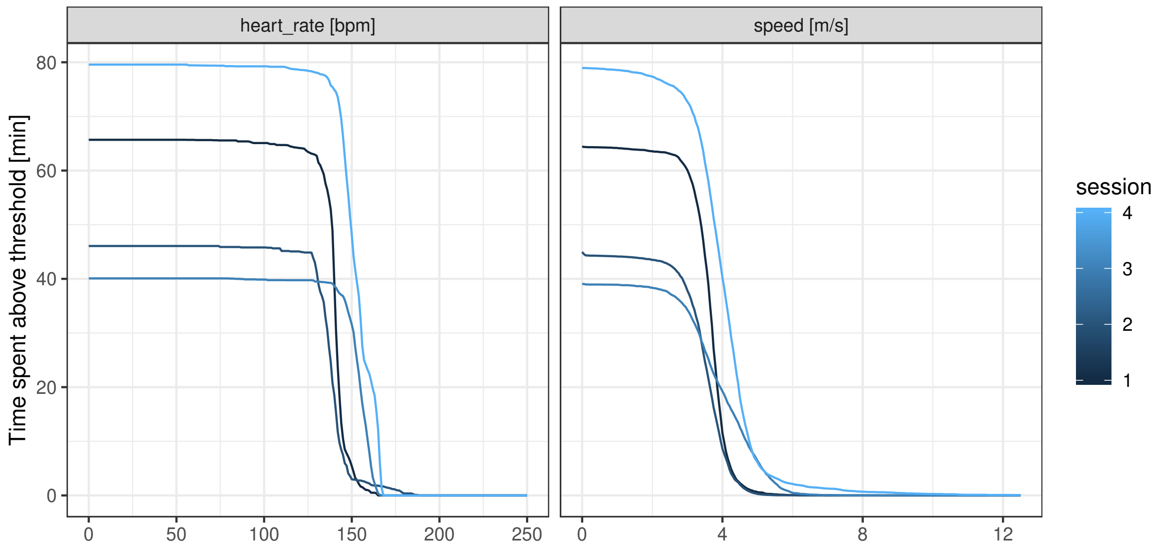

Figure 9: Distribution profiles for sessions 1–4.

Distribution profiles can be calculated using the `distributionProfile()` function which returns an object of class "`distrProfile`". Concentration profiles can be derived from distribution profiles using `concentrationProfile()`, which returns an object of class "`conProfile`". Table 2 includes an overview of constructor functions and available methods for distribution and concentration profiles.

By default, distribution profiles are calculated for speed and heart rate on grids inferred from the data. The following code chunk illustrates the use of `distributionProfile()` and shows how users can specify the variables for which to calculate profiles and the respective grids.

```
R> dProfile <- distributionProfile(runs, session = 1:4,
+    what = c("speed", "heart_rate"),
+    grid = list(speed = seq(0, 12.5, by = 0.05), heart_rate = seq(0, 250)))
R> plot(dProfile, multiple = TRUE)
```

The `multiple` argument of the `plot()` method determines whether to plot the profiles in separate panels (`FALSE`) or overlay them in a common panel (`TRUE`), as in Figure 9. The different session lengths are clearly visible in the height of the curves at 0. Amongst the distribution profiles for speed, the descent of the profile for session 3 is slower than for the other sessions. This difference is most apparent in the concentration profiles, which are shown in Figure 10 and are produced by the following code chunk.

```
R> cProfile <- concentrationProfile(dProfile, what = "speed")
R> plot(cProfile, multiple = TRUE, smooth = TRUE)
```

The profile for session 3 has a mode at around 3.5 meters per second and another one at 5 meters per second, showing that this session involved training at a combination of low and high speeds.

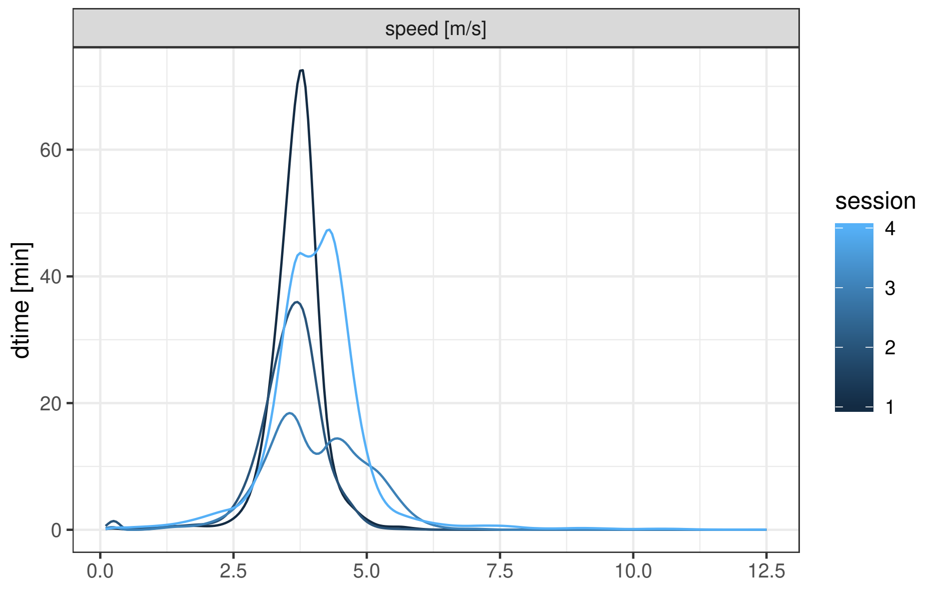

Figure 10: Concentration profiles for sessions 1–4.

6. Handling units of measurement

Data objects of class “`trackeRdata`” and all objects derived from these (“`trackeRdataSummary`”, “`trackeRdataZones`”, “`trackeRWprime`”, “`distrProfile`”, and “`conProfile`”) carry an attribute with the relevant units of measurement. The `getUnits()` method returns the units of measurement for each variable and the `changeUnits()` method can be used to change one or more variables from one set of units to another. The following code chunk displays the current units of `run` for running, changes the unit for speed for running to miles per hour, and displays the changed units.

```
R> subset(getUnits(runs), sport == "running")
```

```
          variable          unit   sport
13        latitude        degree running
14       longitude        degree running
15        altitude             m running
16        distance             m running
17      heart_rate           bpm running
18           speed       m_per_s running
19 cadence_running steps_per_min running
22     temperature             C running
23            pace    min_per_km running
24        duration           min running
```

```
R> runTr2 <- changeUnits(run, variable = "speed", unit = "mi_per_h", sport = "running")
R> subset(getUnits(runTr2), sport == "running")
```

Measurement	Unit(s)
latitude	degrees (degree, default)
longitude	degrees (degree, default)
altitude	meters (m, default), kilometres (km), miles (mi), feet (ft)
distance	meters (m, default), kilometres (km), miles (mi), feet (ft)
speed	meters per second (m_per_s, default), kilometres per hour (km_per_h), feet per minute (ft_per_min), feet per second (ft_per_s), miles per hour (mi_per_h)
cadence_running	steps per minute (steps_per_min, default)
cadence_cycling	revolutions per minute (rev_per_min, default)
power	Watts (W, default), kilowatts (kW)
heart rate	beats per minute (bpm, default)
pace	minutes per kilometre (min_per_km, default), minutes per mile (min_per_mi), seconds per meter (s_per_m)
duration	seconds (s), minutes (min), hours (h) – default is the largest possible unit for which the duration is larger than 1
temperature	degrees Celcius (C, default), degrees Fahrenheit (F)

Table 3: Supported units of measurement.

```
          variable          unit   sport
13        latitude        degree running
14       longitude        degree running
15        altitude             m running
16        distance             m running
17      heart_rate           bpm running
18           speed      mi_per_h running
19 cadence_running steps_per_min running
22     temperature             C running
23            pace    min_per_km running
24        duration           min running
```

Table 3 shows the variables and the corresponding units that are currently supported in **trackeR**.

If objects with different units are `c()`ombined in one object, the units of the first session are applied to all other sessions. Furthermore, the `changeUnits()` method uses name matching to figure out which conversion needs to be done. This allows the user to easily add support for converting from `unitOld` to `unitNew` by authoring a function named `unitOld2unitNew`.

If we wish to report the speed summaries for session 1 in `runSummary` in feet per hour (not currently supported) instead of meters per second, we need to simply provide the appropriately named conversion function as illustrated below. Note that the conversion applies to all speed summaries, i.e., to “average speed” and “average speed moving”.

```
R> m_per_s2ft_per_h <- function(x) x * 3937/1200 * 3600
R> changeUnits(runSummary, variable = "speed", unit = "ft_per_h")
```

```
 *** Session 1 : running ***
```

```
Session times: 2013-06-01 18:32:15 - 2013-06-01 19:37:56
Distance: 14130.7 m
Duration: 65.68 mins
Moving time: 64.17 mins
Average speed: 42349.08 ft_per_h
Average speed moving: 43350.06 ft_per_h
Average pace (per 1 km): 4:38 min:sec
Average pace moving (per 1 km): 4:32 min:sec
Average cadence running: 88.66 steps_per_min
Average cadence cycling: NA rev_per_min
Average cadence running moving: 88.87 steps_per_min
Average cadence cycling moving: NA rev_per_min
Average power: NA W
Average power moving: NA W
Average heart rate: 141.11 bpm
Average heart rate moving: 141.13 bpm
Average heart rate resting: 136.76 bpm
Average temperature: NA C
Total elevation gain: 94.2 m
Work to rest ratio: 42.31

Moving thresholds: 23622 (cycling) 11811 (running)  5906 (swimming) ft_per_h
Unit reference sport: running
```

7. Thresholding and smoothing

There are instances where the data include artefacts due to inaccuracies in the GPS measurements. These can be handled with the `threshold()` method for objects of class "`trackeRdata`", which replaces values outside the specified thresholds with `NA`. The variables and the (lower and upper) thresholds which should be applied for each variable can be specified through the arguments `variable`, `lower`, and `upper`, respectively. An example is given in `?threshold`. The default thresholds are listed in Table 4 and, if necessary, are converted to the units of measurement used for the "`trackeRdata`" object.

The other option for data handling is the `smoother()` method for "`trackeRdata`" objects. This applies a summarising function, such as the mean or median, over a rolling window. Both operations `threshold()` and `smoother()` are used in the `plot()` method for "`trackeRdata`" objects. The default settings for `plot()` are to apply the thresholds specified in Table 4 but not to smooth the data. The top left panel in Figure 11 gives an example where no thresholds are applied and the top right panel uses default settings. The spike to over 20 meters per second in the top left panel is clearly an error in the data; the current world record for 100 meters (by Usain Bolt, August 16, 2009) is 9.58 seconds which translates to an average speed of 10.44 meters per second. The bottom panels show the effect of first applying the default thresholds and then smoothing the data through a rolling median with a window width of 20 observations, either done within the `plot()` method (bottom left) or explicitly via the

Variable	Unit	Lower threshold	Upper threshold	R	C	S
latitude	degrees	-90	90	x	x	x
longitude	degrees	-180	180	x	x	x
altitude	meter	-500	9000	x	x	x
distance	meter	0	∞	x	x	x
heart rate	beats per minute	0	250	x	x	x
speed	meters per second	0	∞	x	x	x
cadence_running	steps per minute	0	∞	x		
cadence_cycling	revolutions per minute	0	∞	x	x	x
power	Watts	0	∞		x	
pace	minutes per kilometre	0	∞	x	x	x
duration	seconds	0	∞	x	x	x
temperature	degrees Celsius	-20	60	x	x	x

Table 4: Default thresholds for running, cycling and swimming data. R stands for running, C stands for cycling and S for swimming, and an x indicates that the corresponding unit applies to that sport.

`threshold()` and `smoother()` methods (bottom right). The following code chunk produces the four plots in Figure 11.

```
R> plot(runs, session = 4, what = "speed", threshold = FALSE)
R> plot(runs, session = 4, what = "speed")
R> plot(runs, session = 4, what = "speed", smooth = TRUE, fun = "median",
+    width = 20)
R> run4 <- threshold(runs[4])
R> run4S <- smoother(run4, what = "speed", fun = "median", width = 20)
R> plot(run4S, what = "speed", smooth = FALSE)
```

The method `smoother()` is also available for distribution and concentration profiles. Smoothing a distribution profile requires a smoothing technique which respects the positivity and monotonicity of the distribution profile. This can be achieved by fitting a shape constrained additive model with Poisson responses as implemented in the **scam** package (Pya 2015). When smoothing concentration profiles, the raw profiles are transformed to distribution profiles which are subsequently smoothed preserving the positivity and monotonicity. The smooth concentration profiles are then derived from the smoothed distribution profiles.

If smooth concentration profiles are all that is required, these can also be computed by using the `concentrationProfile()` method direction on the "`trackeRdata`" object, which will then use an appropriately scaled kernel density estimator on the values of the variables in `what`. This is a more computationally efficient operation, that computing the distribution profiles, smoothing them, and then taking finite differences of the smoothed distribution profiles. The following code chunk illustrates this and plots the concentration profiles using a ridgeline plot (Wilke 2018). The resulting plot is shown in Figure 12.

```
R> cProfile1 <- concentrationProfile(runs,  what = "speed",
+                                    limits = list(speed = c(0, 12.5)))
R> ridges(cProfile1)
```

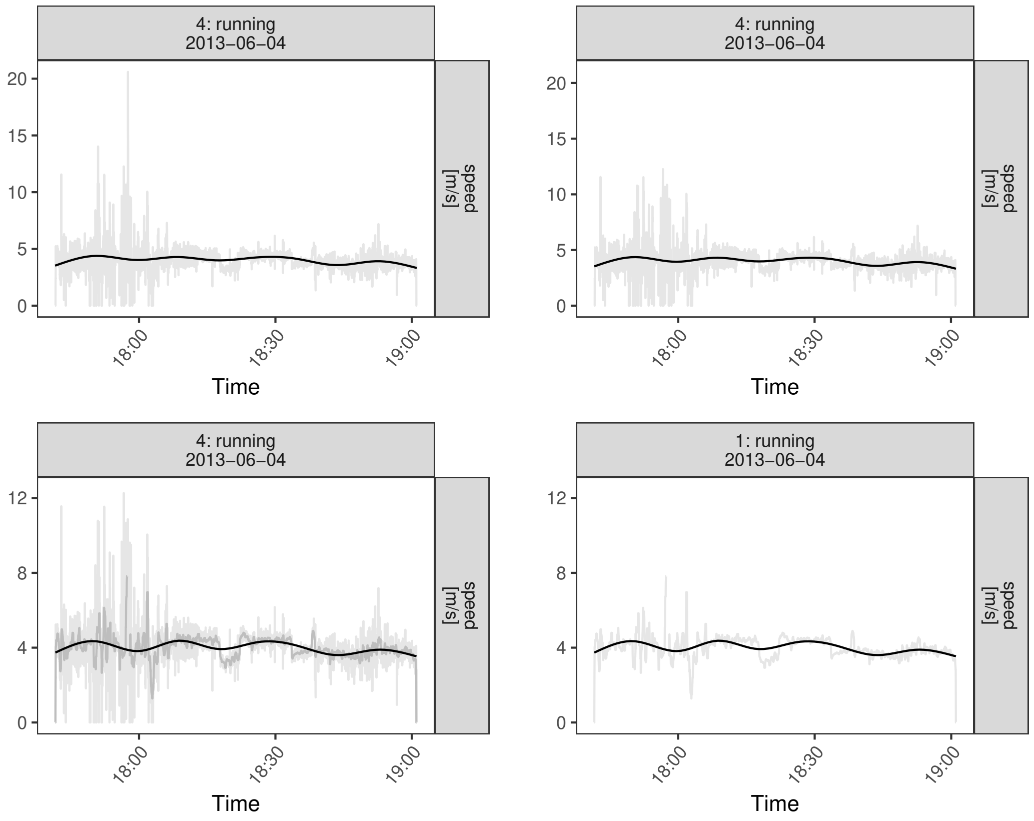

Figure 11: Speed profile of session 4 without thresholding (top left), with the default settings (top right), and with default thresholds as well as smoothing through a rolling median over a window of 20 observations done within the plot function (bottom left) and separately (bottom right).

8. Case study

The example data set included in the package contains 27 sessions of a single male runner in June 2013. A visualisation of scalar summaries for the sessions can be found in Figure 6. The distance covered in those sessions ranges from 2.79 km to 22.35 km, and most sessions were spent moving almost the entire time.

The code chunk below loads the data, applies thresholds, and calculates the smoothed distribution profiles for the 27 sessions. The corresponding concentration profiles are shown in Figure 13.

```
R> library("trackeR")
R> data("runs", package = "trackeR")
R> runsT <- threshold(runs)
R> dpRuns <- distributionProfile(runsT, what = "speed")
R> dpRunsS <- smoother(dpRuns)
R> cpRuns <- concentrationProfile(dpRunsS)
R> plot(cpRuns, multiple = TRUE, smooth = FALSE)
```

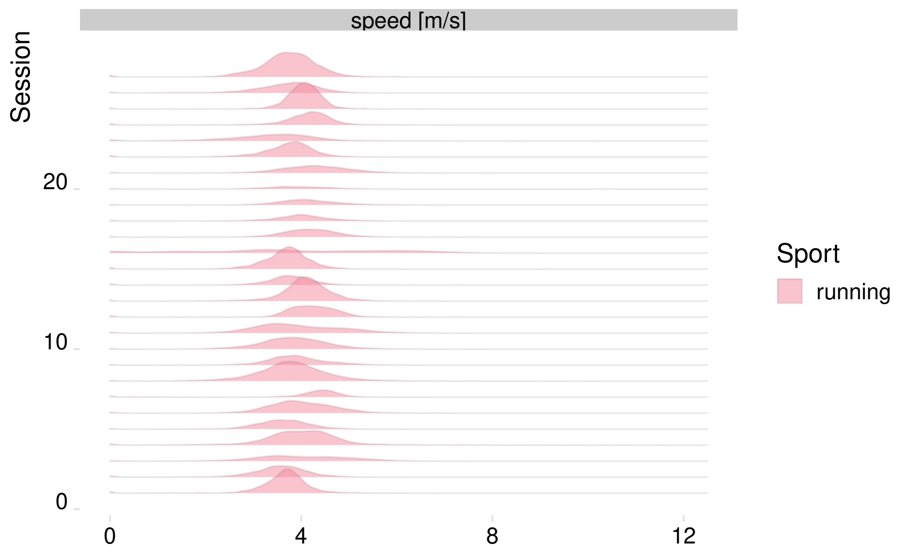

Figure 12: Ridgeline plot of concentration profiles.

The majority of the profiles for speed concentrate around 4 meters per second. However, the curves differ in their shape (unimodal or multimodal), height, and location (revealing concentrations at higher or lower speeds). Functional PCA (e.g., Ramsay and Silverman 2005) can be used to explain those differences ensuring that the profiles are treated directly as functions. **trackeR** contains a convenience function `funPCA()` which converts concentration/distribution profiles to the required functional data format and performs a functional PCA. **trackeR** can also be viewed as a stepping stone to further analysis of tracking data with other R packages. For example, it contains a conversion function, `profile2fd()`, that transforms concentration and distribution profiles to class "`fd`" so that users have direct access to the facilities of the **fda** package (Ramsay, Wickham, Graves, and Hooker 2014) for functional data analysis.

The following code chunk shows the conversion to the required functional data format and the fitting of a functional PCA in separate steps. The PCA has four components and the share of variance is displayed in the last step.

```
R> library("fda")
R> cpFd <- profile2fd(cpRuns, what = "speed")
R> sppca <- pca.fd(cpFd, nharm = 4)
R> varprop <- round(sppca$varprop * 100); names(varprop) <- 1:4
R> varprop
```

```
 1  2  3  4
66 25  6  2
```

The first two harmonics capture 91% of the variation between curves. Since further harmonics capture considerably less variation, only the first two are chosen for further inspection.

Figure 14 shows the mean function (solid line) and the variation captured in the two harmonics (between the dashed and dotted lines). The first harmonic (top panel) illustrates that the

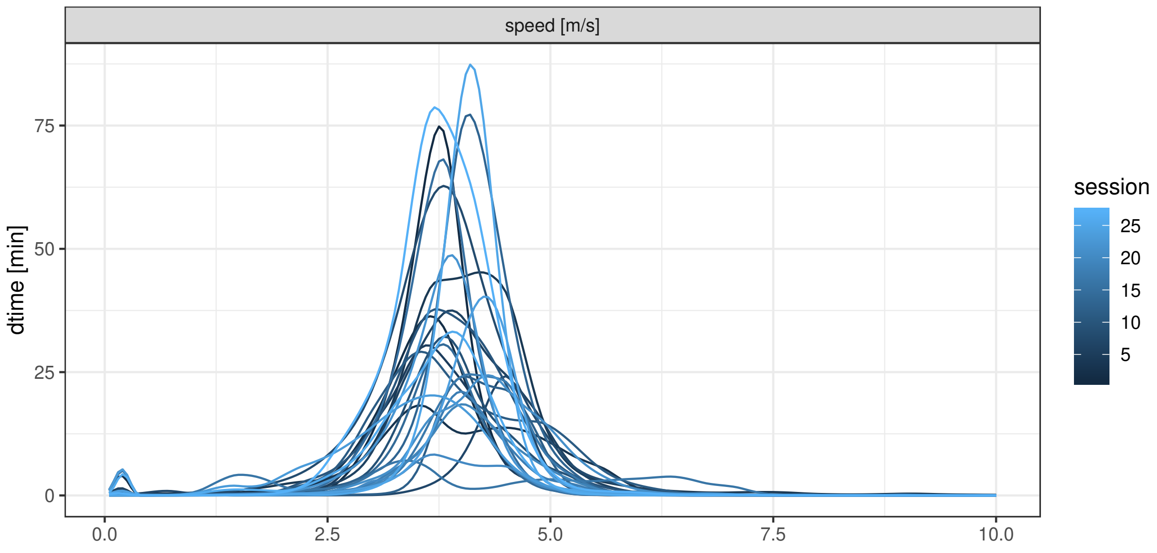

Figure 13: Smoothed speed concentration profiles for all 27 sessions.

most important characteristic of the concentration profiles is the relative value, which is closely related to the overall session duration. The left panel of Figure 15 shows the score on the first harmonic versus "duration moving" which is calculated as part of the scalar session summaries. The second harmonic in the bottom panel of Figure 14 shows variation along the speed thresholds in the centre of the curve. This variation can be explained well by the scalar measure "average speed moving" as shown in the right panel of Figure 15.

The concentration profiles and a functional PCA thus indicate that the two scalar summaries "duration moving" and "average speed moving" provide a good summary of the speed information in the sessions and can be used, for example, in order to incorporate speed as explanatory information in regression analyses.

Acknowledgements

We are thankful to Victoria Downie, Andy Hudson, Louis Passfield, Ben Rosenblatt, and Achim Zeileis for helpful feedback and discussions as well as providing the data that are used for the illustrations and examples in the package.

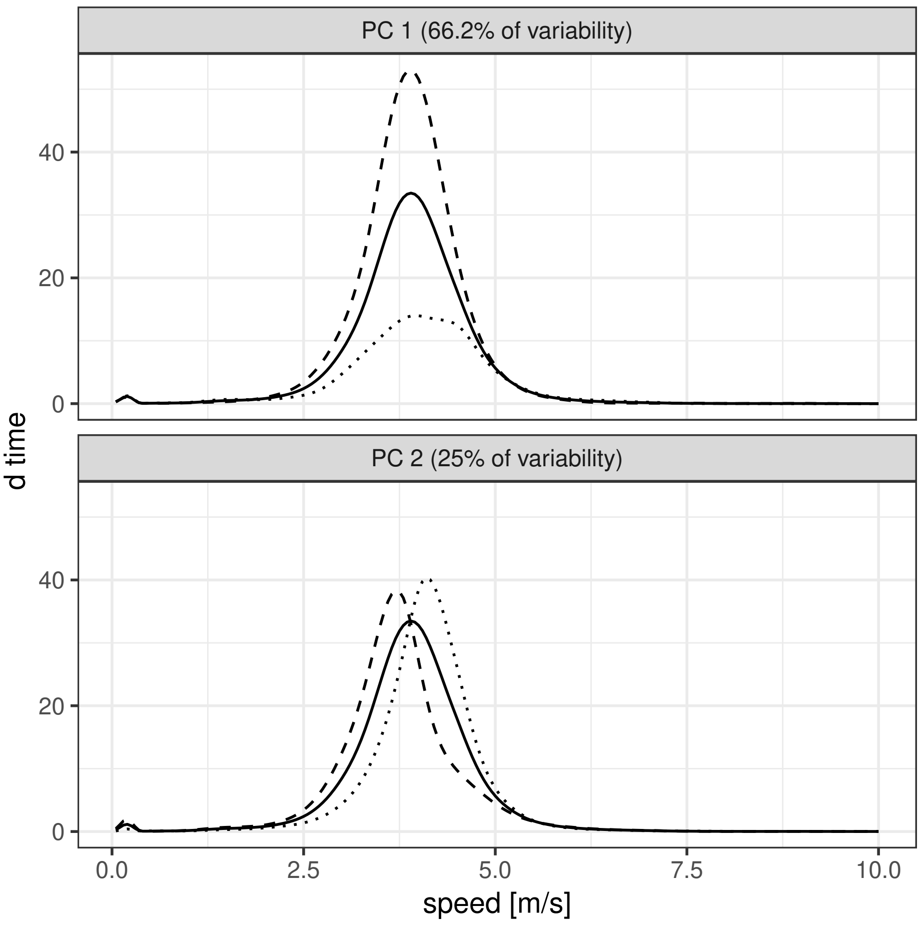

Figure 14: Harmonics 1–2 for the speed concentration profiles. Mean function (solid line) with suitable multiples of the harmonic added (dashed line) and subtracted (dotted line).

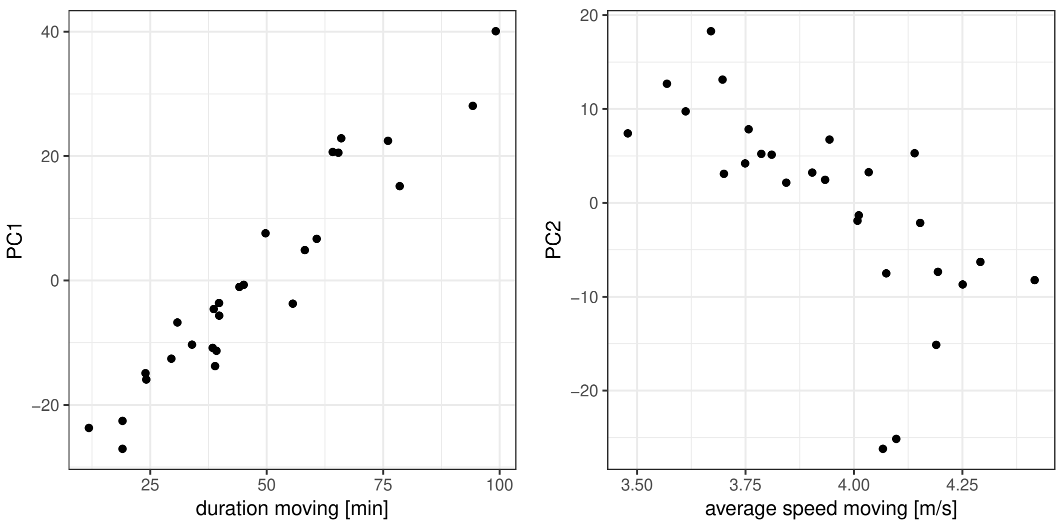

Figure 15: PC1 score vs. "duration moving" (left) and PC2 score vs. "average speed moving" (right).

References

Blume M, Jhirad N, Gassem A (2015). ***pinnacle.API**: A Wrapper for the Pinnacle Sports API.* R package version 1.90, URL `http://CRAN.R-project.org/package=pinnacle.API`.

Bohannon RW (1997). "Comfortable and Maximum Walking Speed of Adults Aged 20–79 Years: Reference Values and Determinants." *Age and Ageing*, **26**(1), 15–19. `doi:10.1093/ageing/26.1.15`.

Calenge C (2006). "The Package **adehabitat** for the R Software: Tool for the Analysis of Space and Habitat Use by Animals." *Ecological Modelling*, **197**(3–4), 516–519. `doi:10.1016/j.ecolmodel.2006.03.017`.

Champely S (2012). ***RcmdrPlugin.SM**: Rcmdr Sport Management Plug-In.* R package version 0.3.1, URL `http://CRAN.R-project.org/package=RcmdrPlugin.SM`.

Cheng J, Xie Y (2016). ***leaflet**: Create Interactive Web Maps with the JavaScript 'Leaflet' Library.* R package version 1.0.1, URL `https://CRAN.R-project.org/package=leaflet`.

Eugster MJA (2013). ***SportsAnalytics**: Infrastructure for Sports Analytics.* R package version 0.1, URL `http://soccer.r-forge.r-project.org/`.

Frick H, Kosmidis I (2017). "**trackeR**: Infrastructure for Running and Cycling Data from GPS-Enabled Tracking Devices in R." *Journal of Statistical Software*, **82**(7), 1–29. `doi:10.18637/jss.v082.i07`.

Garmin Ltd (2013). *Forerunner 310XT Owner's Manual, Rev. G.* URL `http://static.garmincdn.com/pumac/Forerunner310XT_OM_EN.pdf`.

Hijmans RJ (2015). ***raster**: Geographic Data Analysis and Modeling.* R package version 2.4-20, URL `http://CRAN.R-project.org/package=raster`.

Kahle D, Wickham H (2013). "**ggmap**: Spatial Visualization with **ggplot2**." *The R Journal*, **5**(1), 144–161. URL `http://journal.r-project.org/archive/2013-1/kahle-wickham.pdf`.

Karl AT, Broatch J (2015). ***mvglmmRank**: Multivariate Generalized Linear Mixed Models for Ranking Sports Teams.* R package version 1.1-2, URL `http://CRAN.R-project.org/package=mvglmmRank`.

Kosmidis I, Passfield L (2015). "Linking the Performance of Endurance Runners to Training and Physiological Effects via Multi-Resolution Elastic Net." ArXiv e-print arXiv:1506.01388.

Kranstauber B, Smolla M (2015). ***move**: Visualizing and Analyzing Animal Track Data.* R package version 1.5.514, URL `http://CRAN.R-project.org/package=move`.

Mackie J (2015). ***cycleRtools**: Tools for Cycling Data Analysis.* R package version 1.0.4, URL `https://github.com/jmackie4/cycleRtools`.

Monod H, Scherrer J (1965). "The Work Capacity of a Synergic Muscular Group." *Ergonomics*, **8**(3), 329–338. `doi:10.1080/00140136508930810`.

Pebesma E (2012). "**spacetime**: Spatio-Temporal Data in R." *Journal of Statistical Software*, **51**(1). doi:10.18637/jss.v051.i07.

Pebesma E, Klus B (2015). ***trajectories**: Classes and Methods for Trajectory Data*. R package version 0.1-4, URL https://CRAN.R-project.org/package=trajectories.

Pya N (2015). ***scam**: Shape Constrained Additive Models*. R package version 1.1-9, URL http://CRAN.R-project.org/package=scam.

Pya N, Wood SN (2015). "Shape Constrained Additive Models." *Statistics and Computing*, **25**(3), 543–559. doi:10.1007/s11222-013-9448-7.

Ramsay JO, Silverman BW (2005). *Functional Data Analysis*. Springer-Verlag.

Ramsay JO, Wickham H, Graves S, Hooker G (2014). ***fda**: Functional Data Analysis*. R package version 2.4.4, URL http://CRAN.R-project.org/package=fda.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Seiler KS, Kjerland GØ (2006). "Quantifying Training Intensity Distribution in Elite Endurance Athletes: Is there Evidence for an 'Optimal' Distribution?" *Scandinavian Journal of Medicine & Science in Sports*, **16**(1), 49–56. doi:10.1111/j.1600-0838.2004.00418.x.

Skiba PF, Chidnok W, Vanhatalo A, Jones AM (2012). "Modeling the Expenditure and Reconstitution of Work Capacity above Critical Power." *Medicine & Science in Sports & Exercise*, **44**(8), 1526–1532. doi:10.1249/MSS.0b013e3182517a80.

Skiba PF, Fulford J, Clarke DC, Vanhatalo A, Jones AM (2015). "Intramuscular Determinants of the Abilility to Recover Work Capacity above Critical Power." *European Journal of Applied Physiology*, **115**(4), 703–713. doi:10.1007/s00421-014-3050-3.

Sumner MD (2015). ***trip**: Tools for the Analysis of Animal Track Data*. R package version 1.1-21, URL http://CRAN.R-project.org/package=trip.

Wilke CO (2018). ***ggridges**: Ridgeline Plots in 'ggplot2'*. R package version 0.5.0, URL https://CRAN.R-project.org/package=ggridges.

Zeileis A, Grothendieck G (2005). "**zoo**: S3 Infrastructure for Regular and Irregular Time Series." *Journal of Statistical Software*, **14**(6), 1–27. doi:10.18637/jss.v014.i06.

A. Replenishment of W'

Assuming that power is constant, the solution of the differential equation describing the rate of replenishment in Equation 5 with respect to $W'(t)$ gives

$$1 - \frac{W'(t)}{W'_0} = \exp\left(\frac{P - CP}{W'_0} t + D/W'_0\right) . \tag{9}$$

Using Equation 9 over an interval $[t_{i-1}, t_i)$ of constant power gives

$$1 - \frac{W'(t_i)}{W'_0} = \exp\left(\frac{P_i - CP}{W'_0}(t_i - t_{i-1})\right)\left(1 - \frac{W'(t_{i-1})}{W'_0}\right) .$$

Hence, $W'(t_i)$ can be expressed in terms of $W'(t_{i-1})$ as

$$W'(t_i) = W'_0 - \left(W'_0 - W'(t_{i-1})\right) \exp\left(\frac{P_i - CP}{W'_0}(t_i - t_{i-1})\right) .$$

Affiliation:

Hannah Frick
Department of Statistical Science
University College London
Gower Street
London, WC1E 6BT
United Kingdom
E-mail: `hannah.frick@gmail.com`
URL: `http://www.ucl.ac.uk/~ucakhfr/`

Ioannis Kosmidis
Department of Statistics
University of Warwick
Gibbet Hill Road
Coventry, CV4 7AL
United Kingdom
E-mail: `ioannis.kosmidis@warwick.ac.uk`
URL: `http://www.ikosmidis.com`