Package 'walking'

August 22, 2026

Title Segments Accelerometry Data into Walking Bouts using Open Source Methods

Version 0.8.0

Description Segments walking from accelerometry data using 'forest' python module <https://github.com/onnela-lab/forest> from Yi (2025) <doi:10.2196/71375>, 'Verisense' original from Rowlands (2022) <doi:10.1080/02640414.2022.2147134> and 'Verisense' revised from Maylor (2022) <doi:10.3390/s22249984>, and Step Detection Threshold (SDT) from Ducharme (2021) <doi:10.1123/jmpb.2021-0011> methods.

License GPL (>= 3)

Encoding UTF-8

Imports actibase, assertthat, dplyr, lubridate, magrittr, matrixStats, reticulate (>= 1.42.0), signal, tidyr

Suggests readr, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author John Muschelli [aut, cre] (ORCID: <https://orcid.org/0000-0001-6469-1750>)

Maintainer John Muschelli <muschellij2@gmail.com>

Repository CRAN

Date/Publication 2026-08-22 15:50:02 UTC

Contents

`estimate_steps_forest` *Estimate Steps with Unified Syntax*

Description

Estimate Steps with Unified Syntax

Usage

```
estimate_steps_forest(data, ...)

have_forest()

estimate_steps_verisense(
  data,
  resample_to_15hz = FALSE,
  ...,
  method = c("original", "revised")
)

estimate_steps_sdt(data, ...)
```

Arguments

`data`	A `data.frame` with a column for time in `POSIXct` (usually `HEADER_TIMESTAMP`), and `X`, `Y`, `Z`
`...`	additional arguments to pass to specific method
`resample_to_15hz`	resample the data to 15Hz as `verisense` indicated that was the sample rate this was designed under.
`method`	Either using the original parameters `verisense_count_steps()` or the revised parameters `verisense_count_steps_revised()`

Value

A `data.frame` of seconds and steps

Examples

```
run_example = FALSE
csv_file = system.file("test_data_bout.csv", package = "walking")
if (requireNamespace("readr", quietly = TRUE)) {
  run_example = TRUE
  x = readr::read_csv(csv_file)
  colnames(x)[colnames(x) == "UTC time"] = "time"
  out = estimate_steps_verisense(x, sample_rate = 10L,
                                 method = "original")
```

```
  out = estimate_steps_verisense(x, sample_rate = 10L,
                                 method = "revised")
  out = estimate_steps_sdt(x, sample_rate = 10L)
}

if (requireNamespace("readr", quietly = TRUE) &&
    reticulate::py_module_available("forest")) {
  out = estimate_steps_forest(x, sample_rate_analysis = 10L)
}
```

`find_walking` *Finds walking and calculate steps from raw acceleration data.*

Description

Method finds periods of repetitive and continuous oscillations with predominant frequency occurring within know step frequency range. Frequency components are extracted with Continuous Wavelet Transform.

Usage

```
find_walking(
  data,
  sample_rate_analysis = 10L,
  min_amplitude = 0.3,
  step_frequency = c(1.4, 2.3),
  alpha = 0.6,
  beta = 2.5,
  min_duration_peak = 3L,
  delta = 20L,
  verbose = TRUE,
  disable_parellelization = TRUE
)
```

Arguments

`data`	A `data.frame` with a column for time in `POSIXct` (usually `HEADER_TIMESTAMP`), and `X`, `Y`, `Z`
`sample_rate_analysis`	sampling frequency (in Hz) **for analyzing walking**. Note, this is **NOT** the sampling frequency of the data.
`min_amplitude`	minimum amplitude (in g)
`step_frequency`	step frequency range
`alpha`	maximum ratio between dominant peak below and within step frequency range
`beta`	maximum ratio between dominant peak above and within step frequency range

`min_duration_peak`
: minimum duration of peaks (in seconds)

`delta`
: maximum difference between consecutive peaks (in multiplication of 0.05Hz)

`verbose`
: print diagnostic messages

`disable_parellelization`
: disable any back-end parallelization that is done in oak that is from `ssqueezepy`. See https://github.com/OverLordGoldDragon/ssqueezepy#gpu--cpu-acceleration

Value

A vector of number of steps per second

Examples

```
 csv_file = system.file("test_data_bout.csv", package = "walking")
if (requireNamespace("readr", quietly = TRUE) && reticulate::py_module_available("forest")) {
  x = readr::read_csv(csv_file)
  colnames(x)[colnames(x) == "UTC time"] = "time"
  res = find_walking(data = x)
 }
```

`preprocess_bout` *Preprocesses accelerometer bout to a common format.*

Description

Resample 3-axial input signal to a predefined sampling rate and compute vector magnitude.

Usage

```
preprocess_bout(data, sample_rate = 10L)

preprocess_bout_r(data, sample_rate = 10L)
```

Arguments

`data`
: A `data.frame` with a column for time in `POSIXct` (usually `HEADER_TIMESTAMP`), and `X`, `Y`, `Z`

`sample_rate`
: sampling frequency, coercible to an integer. This is the sampling rate you're sampling the data *into*.

Value

A list of `vm_bout`, which is an unnamed list of length 2, containing times and VM (vector magnitude) interpolated. This will be passed into other `forest` functions. The list also contains `vm_data`, which transforms `vm_bout` into a `data.frame` after creating the required times.

Examples

```
csv_file = system.file("test_data_bout.csv", package = "walking")
if (requireNamespace("readr", quietly = TRUE)) {
  x = readr::read_csv(csv_file)
  colnames(x)[colnames(x) == "UTC time"] = "time"
  if (reticulate::py_module_available("forest")) {
    res = preprocess_bout(data = x)
    res2 = preprocess_bout_r(data = x)
    testthat::expect_equal(res$vm_bout, res2$vm_bout, tolerance = 1e-4)
  }
}
```

---

py_require_forest *Require Forest module*

Description

Require Forest module

Usage

```
py_require_forest(python_version = "3.11", ...)
```

Arguments

python_version	Python version to use, passed to reticulate::py_require()
...	additional arguments passed to reticulate::py_require()

Value

invisible NULL

Examples

```
files <- list.files()
py_require_forest()
walking:::cleanup_uv_lock_files()
stopifnot(length(setdiff(list.files(), files)) == 0L)
```

sdt_count_steps *Estimate Steps via SDT*

Description

Estimate Steps via SDT

Usage

```
sdt_count_steps(
  data,
  sample_rate,
  order = 4L,
  high = 0.25,
  low = 2.5,
  location = c("wrist", "waist"),
  verbose = TRUE
)
```

Arguments

data	A data.frame with a column for time in POSIXct (usually HEADER_TIMESTAMP), and X, Y, Z
sample_rate	sampling frequency (in Hz)
order	order of bandpass Butterworth filter order
high	highpass threshold for filter
low	lowpass threshold for filter
location	location of the device, which indicates a different threshold for peaks
verbose	print diagnostic messages

Value

A data.frame of timeand steps

Examples

```
csv_file = system.file("test_data_bout.csv", package = "walking")
if (requireNamespace("readr", quietly = TRUE)) {
  x = readr::read_csv(csv_file)
  colnames(x)[colnames(x) == "UTC time"] = "time"
  res = sdt_count_steps(data = x, sample_rate = 100L)
}
```

`verisense_count_steps` *Count Steps According to Gu et al, 2017 Method*

Description

This method is based off finding peaks in the summed and squared acceleration signal and then using multiple thresholds to determine if each peak is a step or an artifact. An additional magnitude threshold was added to the algorithm to prevent false positives in free living data.

Usage

```
verisense_count_steps(
  data,
  sample_rate,
  k = 3,
  periodicity_range = c(5, 15),
  similarity_threshold = -0.5,
  continuity_window_size = 4,
  continuity_threshold = 4,
  variance_threshold = 0.001,
  vm_threshold = 1.2,
  peak_finder = c("fast", "original"),
  verbose = TRUE,
  global_vm_threshold = 0.025
)

verisense_count_steps_revised(
  ...,
  k = 4,
  periodicity_range = c(4, 20),
  similarity_threshold = -1,
  continuity_window_size = 4,
  continuity_threshold = 4,
  variance_threshold = 0.01,
  vm_threshold = 1.25
)
```

Arguments

`data`	A `data.frame` with a column for time in `POSIXct` (usually `HEADER_TIMESTAMP`, not required), and `X`, `Y`, `Z`
`sample_rate`	sampling frequency of the input data
`k`	window size for controlling peak finding.
`periodicity_range`	a length-2 vector of the range of periodicity. These are integers that represent **samples** not seconds.

`similarity_threshold`
: threshold (in g) for similarity between magnitude of peaks

`continuity_window_size`
: Window size for continuity

`continuity_threshold`
: Threshold for continuity

`variance_threshold`
: Variance threshold for the signal

`vm_threshold`
: vector magnitude threshold for a peak to be called a peak

`peak_finder`
: function to find peaks, either the "original" from the code, or the optimized "fast" version.

`verbose`
: print diagnostic messages

`global_vm_threshold`
: Global acceleration VM threshold (in standard deviation) for the total vector. If `sd(acc) < thresh` no steps are estimated. Set to `0` to run estimation regardless.

`...`
: not used, used to passes arguments from `verisense_count_steps_revised` to `verisense_count_steps`

Value

A vector of length `round(nrow(input_data) / sample_rate)` of the estimated steps, where the data is rounded to seconds

Note

the `_revised` version is the same algorithm with different defaults for the parameters as based on doi:10.3390/s22249984.

Author(s)

Matthew R Patterson mpatterson@shimmersensing.com, MIT license, Copyright (c) 2020 Shimmer

Examples

```
csv_file = system.file("test_data_bout.csv", package = "walking")
if (requireNamespace("readr", quietly = TRUE)) {
  x = readr::read_csv(csv_file)
  colnames(x)[colnames(x) == "UTC time"] = "time"
  out = verisense_count_steps(x, sample_rate = 10L)
}
input_data <- matrix(runif(500 * 3, min = -1.5, max = 1.5), ncol = 3)
verisense_count_steps(input_data, sample_rate = 15L)
verisense_count_steps(input_data, sample_rate = 15L, peak_finder = "fast")
acc = sqrt(rowSums(input_data^2))
verisense_count_steps(acc, sample_rate = 15L, peak_finder = "fast")
```

Index