

Package ‘walking’

August 25, 2026

Title Segments Accelerometry Data into Walking Bouts using Open Source Methods

Version 0.8.2

Description Segments walking from accelerometry data using 'forest' python module <<https://github.com/onnella-lab/forest>> from Yi (2025) <[doi:10.2196/71375](https://doi.org/10.2196/71375)>, 'Verisense' original from Rowlands (2022) <[doi:10.1080/02640414.2022.2147134](https://doi.org/10.1080/02640414.2022.2147134)> and 'Verisense' revised from Maylor (2022) <[doi:10.3390/s22249984](https://doi.org/10.3390/s22249984)>, and Step Detection Threshold (SDT) from Ducharme (2021) <[doi:10.1123/jmpb.2021-0011](https://doi.org/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 <muschelli2@gmail.com>

Repository CRAN

Date/Publication 2026-08-25 11:00:02 UTC

Contents

estimate_steps_forest	2
find_walking	3
preprocess_bout	4
py_require_forest	5
sdt_count_steps	6
verisense_count_steps	7

Index	9
--------------	----------


```

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

  Sys.setenv("SSQ_PARALLEL" = 0)
  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_parallelization = 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_parallelization	Whether to disable the CPU parallelization used by oak through ssqueezepy. Parallelization can speed up larger analyses but may conflict with an R/Python environment or its threading and OpenMP configuration. The default, TRUE, sets SSQ_PARALLEL=0 while this function runs. If forest fails to load, set Sys.setenv(SSQ_PARALLEL = 0) before calling <code>py_require_forest()</code> and <code>reticulate::import("forest")</code> . See https://github.com/OverLordGoldDragon/ssqueezepy#gpu--cpu-acceleration .

Value

A vector of number of steps per second

Examples

```
Sys.setenv("SSQ_PARALLEL" = 0)
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	<i>Preprocesses accelerometer bout to a common format.</i>
-----------------	--

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 <code>data.frame</code> with a column for time in POSIXct (usually <code>HEADER_TIMESTAMP</code>), and X, Y, Z
sample_rate	sampling frequency, coercible to an integer. This is the sampling rate you're sampling the data <i>into</i> .

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	<i>Require Forest module</i>
-------------------	------------------------------

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	<i>Estimate Steps via SDT</i>
-----------------	-------------------------------

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 time and 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 <code>verisense_count_steps_revised</code> to <code>verisense_count_steps</code>

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](https://doi.org/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

`estimate_steps_forest`, [2](#)
`estimate_steps_sdt`
 (`estimate_steps_forest`), [2](#)
`estimate_steps_verisense`
 (`estimate_steps_forest`), [2](#)

`find_walking`, [3](#)

`have_forest` (`estimate_steps_forest`), [2](#)

`preprocess_bout`, [4](#)
`preprocess_bout_r` (`preprocess_bout`), [4](#)
`py_require_forest`, [5](#)
`py_require_forest()`, [4](#)

`reticulate::py_require()`, [5](#)

`sdt_count_steps`, [6](#)

`verisense_count_steps`, [7](#)
`verisense_count_steps()`, [2](#)
`verisense_count_steps_revised`
 (`verisense_count_steps`), [7](#)
`verisense_count_steps_revised()`, [2](#)