Package ‘accelerometry’

May 12, 2015

Type Package
Title Functions for Processing Minute-to-Minute Accelerometer Data
Version 2.2.5
Date 2015-05-11
Author Dane R. Van Domelen
Maintainer Dane R. Van Domelen <vandomed@gmail.com>
Description A collection of functions that perform operations on time-series accelerometer data, such as identify non-wear time, flag minutes that are part of an activity bout, and find the maximum 10-minute average count value. The functions are generally very flexible, allowing for a variety of algorithms to be implemented. Most of the functions are written in C++ for efficiency.
License GPL-2
Depends R (>= 3.0.0)
Imports Rcpp (>= 0.11.0)
LinkingTo Rcpp
Repository CRAN
Repository/R-Forge/Project accelerometry
Repository/R-Forge/Revision 82
Repository/R-Forge/DateTimeStamp 2015-05-11 18:15:35
Date/Publication 2015-05-12 18:06:59
NeedsCompilation yes

R topics documented:

  accelerometry-package .................................................. 2
  accel.artifacts ......................................................... 4
  accel.bouts ............................................................ 5
  accel.intensities ....................................................... 7
  accel.process.tri ....................................................... 8
  accel.process.uni ....................................................... 13
Description

A collection of functions that perform operations on time-series accelerometer data, such as identify non-wear time, flag minutes that are part of an activity bout, and find the maximum 10-minute average count value. The functions are generally very flexible, allowing for a variety of algorithms to be implemented. Most of the functions are written in C++ for efficiency.

Details

Package: accelerometer
Type: Package
Version: 2.2.5
Date: 2015-05-11
License: GPL-2

The following functions are included in the package:

accel.artifacts, accel.bouts, accel.process.uni, accel.process.tri, accel.intensities, accel.sedbreaks, accel.weartime, blockaves, movingaves, rle2, inverse.rle2

Most of these functions use C++ code, added via the 'Rcpp' package [1, 2].

Two datasets are included for the Examples: unidata and tridata. unidata contains uniaxial data on the first five participants in NHANES 2003-2004 [3], and tridata contains seven days of triaxial data from a volunteer, provided by Ei Ei Khaing Nang from the Department of Epidemiology and Public Health, Yong Loo Lin School of Medicine, National University of Singapore, Singapore, Republic of Singapore.

Some additional information on the package ‘accelerometry’ and its functions can be found on Dane’s website, https://sites.google.com/site/danevandomelen/
Note

The package ‘nhanesaccel’ has functions specifically for processing data from the National Health and Nutrition Examination Survey (NHANES), years 2003-2006. Users who wish to process NHANES data can install ‘nhanesaccel’ from R-Forge [4].

Author(s)

Dane R. Van Domelen
Maintainer: Dane R. Van Domelen <vandomed@gmail.com>

References


Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.

Examples

```
# Load in sample data from NHANES 2003-2004 [3]
data(unidata)

# Get data from ID number 21005
id.part1 <- unidata[unidata[, "seqn"] == 21005, "seqn"]
days.part1 <- unidata[unidata[, "seqn"] == 21005, "paxday"]
counts.part1 <- unidata[unidata[, "seqn"] == 21005, "paxinten"]

# Identify periods of valid wear time
weartime.flag <- accel.weartime(counts = counts.part1)

# Flag minutes that are part of a moderate-to-vigorous activity bout
mvpa.bouts.flag <- accel.bouts(counts = counts.part1, weartime = weartime.flag, thresh.lower = 2020)

# Obtain maximum 10-minute count average
max.10min.movingave <- movingaves(x = counts.part1, window = 10, return.max = TRUE)
```

# Process data from ID 21005 and request per-day variables and daily averages
accel.list <- accel.process.uni(counts = counts.part1, id = id.part1, return.form = 3)

# Process data according to methods used in NCI's SAS programs [5]
accel.nci <- accel.process.uni(counts = counts.part1, id = id.part1, nci.methods = TRUE, 
brevity = 2, return.form = 3)

# Load in triaxial sample data
data(tridata)

# Process data and request per-day variables
accel.days <- accel.process.tri(counts = tridata[, 1:3], steps = tridata[, 4])

# Process data, but for non-wear detection use triaxial vector magnitude with 90-
# minute window and two-minute tolerance for nonzero counts up to 200
accel.days <- accel.process.tri(counts = tridata[, 1:3], steps = tridata[, 4],
nonwear.axis = "mag", nonwear.window = 90,
nonwear.tol = 2, nonwear.tol.upper = 200)

---

accel.artifacts Accelerometer Artifact Correction

### Description

This function corrects abnormally high count values in minute-to-minute accelerometer data by replacing such values with the average of the neighboring count values.

### Usage

```r
accel.artifacts(counts, thresh = 32767, skipchecks = FALSE)
```

### Arguments

- `counts` Time series accelerometer counts vector.
- `thresh` Minimum count value that is considered an artifact.
- `skipchecks` If TRUE, function skips error checking code and runs slightly faster.

### Value

An integer vector identical to the input vector counts but with artifacts corrected.

### Note

An integer vector is returned despite the average calculation often producing a decimal. This follows the convention used by the NCI’s SAS programs [1].

Some additional information on the package accelerometry and its functions can be found on the author’s website, [https://sites.google.com/site/danevandomelen/](https://sites.google.com/site/danevandomelen/)
Author(s)
Dane R. Van Domelen

References

Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.

See Also
accel.process.uni, accel.process.tri

Examples

```r
# Load in sample data frame
data(unidata)

# Get data from ID number 21007
counts.part3 <- unidata[, unidata[, "seqn"] == 21007, "paxinten"]

# Replace artifacts (defined as 10000+ counts) with average of neighboring values
counts.part3.corrected <- accel.artifacts(counts = counts.part3, thresh = 10000)
```

### accel.bouts

**Activity Bout Detection**

This function identifies bouts of physical activity in minute-to-minute accelerometer data.

#### Usage

```r
accel.bouts(counts, weartime = NULL, bout.length = 10, thresh.lower = 0, thresh.upper = Inf, tol = 0, tol.lower = 0, tol.upper = Inf, nci = FALSE, days.distinct = FALSE, skipchecks = FALSE)
```

#### Arguments

- **counts**: Time series accelerometer counts vector.
- **weartime**: Accelerometer wear time vector; must be same length as counts and consist of 1’s and 0’s (if specified).
- **bout.length**: Minimum length of an activity bout.
- **thresh.lower**: Lower cut-off for count values in intensity range.
### thresh.upper
Upper cut-off for count values in intensity range.

### tol
Number of minutes with count values outside of intensity range allowed during an activity bout.

### tol.lower
Lower cut-off for count values outside of intensity range during an activity bout.

### tol.upper
Upper cut-off for count values outside of intensity range during an activity bout.

### nci
If TRUE, use activity bouts algorithm from the NCI’s SAS programs [1]; if FALSE, use regular algorithm (see Details).

### days.distinct
If TRUE, treat each day of data as distinct, i.e. identify non-wear time and activity bouts in day 1, then day 2, etc.; If FALSE, apply algorithms on continuous basis for full monitoring period. If protocol has participants remove accelerometer for sleep, strongly recommend setting to FALSE to capture non-wear periods that start between 11 pm and midnight. Function assumes that first 1440 data points are day 1, next 1440 are day 2, and so on.

### skipchecks
If TRUE, function skips error checking code and runs slightly faster.

### Details
If nci is set to FALSE, the algorithm uses a moving window to go through every possible interval of length bout.length in input vector counts. Any interval in which all counts are greater than or equal to tol.lower and less than or equal to tol.upper, and no more than tol counts are less than thresh.lower or greater than thresh.upper, is classified as an activity bout.

If nci is set to TRUE, activity bouts are classified according to the algorithm used in the NCI’s SAS programs [1]. Briefly, this algorithm defines an activity bout as an interval of length bout.length that starts with a count value between thresh.lower and thresh.upper and has no more than tol counts outside of that range. If these criteria are met, the bout continues until there are (tol + 1) consecutive minutes outside of the range defined by thresh.lower and thresh.upper. The parameters tol.lower and tol.upper are not used.

If the user allows for a tolerance (e.g. tol = 2) and does not use the NCI algorithm (nci = FALSE), specifying a non-zero value for tol.lower is highly recommended. Otherwise the algorithm will tend to classify minutes immediately before and after an activity bout as being part of the bout.

### Value
Integer vector of same length as counts and weartime, with 1’s indicating minutes that are part of an activity bout, and 0’s indicating minutes that are not part of an activity bout.

### Note
Specifying thresh.lower but leaving the default (Inf) for thresh.upper is generally recommended. Specifying both of these parameters can be overly restrictive in that the algorithm may miss bouts of activity in which counts are consistently high, but not exclusively in one intensity range.

Some additional information on the package accelerometry and its functions can be found on the author’s website, [https://sites.google.com/site/danevandomelen/](https://sites.google.com/site/danevandomelen/)

### Author(s)
Dane R. Van Domelen
References


Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.

See Also

accel.weartime, accel.process.uni, accel.process.tri

Examples

```r
# Load in sample data frame
data(unidata)

# Get data from ID number 21005
counts.part1 <- unidata[unidata[, "seqn"] == 21005, "paxinten"]

# Identify periods of valid wear time
weartime.flag <- accel.weartime(counts = counts.part1)

# Flag minutes that are part of a moderate-to-vigorous activity bout
mvpa.bouts.flag <- accel.bouts(counts = counts.part1, weartime = weartime.flag,
                               thresh.lower = 2020)
```

### accel.intensities

**Classification of Physical Activity Intensities**

**Description**

This function computes the number of minutes with counts in user-defined intensity levels, and the number of counts accumulated during time spent in each intensity level. It is intended for use with minute-to-minute accelerometer data.

**Usage**

```r
accel.intensities(counts, thresh = c(100, 760, 2020, 5999), skipchecks = FALSE)
```

**Arguments**

- **counts**: Time series accelerometer counts vector.
- **thresh**: Vector of four cut-points from which five intensity ranges are derived. For example, if `thresh = c(100, 760, 2020, 5999)`, minutes with 0-99 counts are classified as intensity level 1, minutes with 100-759 counts are classified as intensity level 2, ..., and minutes with 5999 or greater counts are classified as intensity level 5. By default the function also returns time in intensities 2-3, 4-5, and 2-5.
- **skipchecks**: If TRUE, function skips error checking code and runs slightly faster.
Value
An integer vector in which the first eight values are minutes in intensity 1, 2, 3, 4, 5, 2-3, 4-5, and 2-5, respectively, and the next eight values are counts accumulated during time spent in intensity 1, 2, 3, 4, 5, 2-3, 4-5, and 2-5. Intensities 1-5 typically correspond to sedentary, light, lifestyle, moderate, and vigorous.

Note
Users should generally input a vector of counts that occur during wear time only. Otherwise the estimate for intensity range 1 (sedentary) will include non-wear time and be vastly overestimated. Some additional information on the package accelerometry and its functions can be found on the author's website, https://sites.google.com/site/danevandomelen/

Author(s)
Dane R. Van Domelen

References
Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.

See Also
accel.weartime, accel.process.uni, accel.process.tri

Examples
# Load in sample data frame
data(unidata)

# Get data from ID number 21005
counts.part1 <- unidata[, "seqn"] == 21005, "paxinten"]

# Create vector of counts during valid wear time only
counts.part1.weartime <- counts.part1[accel.weartime(counts = counts.part1) == 1]

# Calculate physical activity intensity variables
intensity.variables <- accel.intensities(counts = counts.part1.weartime)

Description
This function calculates a variety of physical activity variables based on triaxial minute-to-minute accelerometer data for individual participants. A data dictionary for the variables returned is available here: https://sites.google.com/site/danevandomelen/r-package-accelerometry/data-dictionary.
Usage

```r
accel.process.tri(counts.tri, steps = NULL, nci.methods = FALSE,
start.date = as.Date("2014/1/5"), start.time = "00:00:00", id = NULL,
brevity = 1, valid.days = 1, valid.week.days = 0,
valid.weekend.days = 0, int.axis = "vert",
int.cuts = c(100, 760, 2020, 5999), cpm.nci = FALSE,
hourly.axis = "vert", days.distinct = FALSE, nonwear.axis = "vert",
nonwear.window = 60, nonwear.tol = 0, nonwear.tol.upper = 99,
nonwear.nci = FALSE, weartime.minimum = 600, weartime.maximum = 1440,
partialday.minimum = 1440, active.bout.length = 10,
active.bout.tol = 0, mvpa.bout.tol.lower = 0, vig.bout.tol.lower = 0,
active.bout.nci = FALSE, sed.bout.tol = 0,
artifact.thresh = 25000, artifact.action = 1, weekday.weekend = FALSE,
return.form = 2)
```

Arguments

- **counts.tri**: Three-column accelerometer counts matrix or data frame, where columns 1-3 represent vertical, anteroposterior (AP), and mediolateral (ML) counts, respectively.
- **steps**: Steps vector. If specified, must be same length as counts.tri.
- **nci.methods**: If TRUE, inputs are set to replicate the data processing methods used by the NCI's SAS programs [1]. More specifically:
  - valid.days = 4; valid.week.days = 0; valid.weekend.days = 0; int.axis = "vert";
  - int.cuts = c(100, 760, 2020, 5999); cpm.nci = TRUE; hourly.axis = "vert";
  - days.distinct = TRUE; nonwear.axis = "vert"; nonwear.window = 60; nonwear.tol = 2;
  - nonwear.tol.upper = 100; nonwear.nci = TRUE; weartime.minimum = 600;
  - weartime.maximum = 1440; partialday.minimum = 1440; active.bout.length = 10;
  - active.bout.tol = 2; mvpa.bout.tol.lower = 0; vig.bout.tol.lower = 0;
  - active.bout.nci = TRUE; sed.bout.tol = 0; sed.bout.tol.maximum = 759;
  - artifact.axis = "vert"; artifact.thresh = 32767; artifact.action = 3

  Even if nci.methods is set to TRUE, the user can specify non-default values for brevity, weekday.weekend, and return.form.
- **start.date**: Date of first day of monitoring (must be of class ‘date’). Only used to extract day of week, so if day of week is known but date is not, user can enter any date that corresponds to that day of the week. The default date corresponds to the first Sunday in January 2014.
- **start.time**: Optional character vector indicating the start time for monitoring. If not specified it is assumed to be 00:00:00, i.e. the very beginning of the first day.
- **id**: Either a single value or a vector indicating the ID number for the participant whose accelerometer data was entered.
- **brevity**: Controls the number of physical activity variables returned. If 1, returns basic indicators of physical activity volume; if 2, also returns indicators of activity intensities, activity bouts, sedentary behavior, and peak activity; if 3, also returns hourly count averages.
valid.days Minimum number of valid days to be considered valid for analysis.
valid.weekdays Minimum number of valid weekdays to be considered valid for analysis.
valid.weekend.days Minimum number of valid weekend days to be considered valid for analysis.
int.axis Axis that should be used to classify intensities. Should be one of "vert", "ap", "ml", "sum" (for triaxial sum), or "mag" (for triaxial vector magnitude).
int.cuts Vector of four cut-points from which five intensity ranges are derived using the accelerometer axis specified by int.axis. For example, if int.axis = "vert" and thresh = c(100, 760, 2020, 5999), minutes with 0-99 vertical axis counts are classified as intensity level 1, minutes with 100-759 counts are classified as intensity level 2, ... , and minutes with 5999 or greater counts are classified as intensity level 5. Intensities 1-5 typically correspond to sedentary, light, lifestyle, moderate, and vigorous.
cpm.nci If TRUE, average counts per minute is calculated by dividing average daily counts by average daily weartime, as opposed to averaging each day's counts per minute value. In general, leave as FALSE unless you want to replicate the NCI's SAS programs [1].
hourly.axis Axis that should be used for hourly counts per minute variables. Should be one of "vert", "ap", "ml", "sum" (for triaxial sum), or "mag" (for triaxial vector magnitude).
days.distinct If TRUE, treat each day of data as distinct, i.e. identify non-wear time and activity bouts in day 1, then day 2, etc.: If FALSE, apply algorithms on continuous basis for full monitoring period. If protocol has participants remove accelerometer for sleep, strongly recommend setting to FALSE to capture non-wear periods that start between 11 pm and midnight.
nonwear.axis Axis that should be used for non-wear algorithm. Should be one of "vert", "ap", "ml", "sum" (for triaxial sum), or "mag" (for triaxial vector magnitude).
nonwear.window Minimum length of a non-wear interval.
nonwear.tol Number of minutes with non-zero counts allowed during a non-wear interval.
nonwear.tol.upper Maximum count value for a minute with non-zero counts during a non-wear interval.
nonwear.nci If TRUE, use non-wear algorithm from the NCI's SAS programs [1]; if FALSE, use regular algorithm (see Details).
weartime.minimum Minimum number of wear time minutes for a day of monitoring to be considered valid.
weartime.maximum Maximum number of wear time minutes for a day of monitoring to be considered valid.
partialday.minimum Minimum number of minutes for a partial day of monitoring to be processed and potentially considered valid for analysis (generally applies only to the first and last days of monitoring, which may not cover full 24-hour periods). This input
is included because some researchers may prefer to exclude a day that only has data from, say, 1 pm to midnight. Even though there may be sufficient wear time during that period to be classified as a valid day, the missing chunk of data prior to 1 pm may result in the day not being representative of the participant's usual physical activity.

- **active.bout.length**: Minimum length of moderate-to-vigorous physical activity (MVPA) and vigorous physical activity (VPA) bouts.
- **active.bout.tol**: Number of minutes with counts below the required intensity level allowed during MVPA and VPA bouts.
- **mvpa.bout.tol.lower**: Lower cut-off for count values outside of MVPA intensity range during an MVPA bout.
- **vig.bout.tol.lower**: Lower cut-off for count values outside of VPA intensity range during a VPA bout.
- **active.bout.nci**: If TRUE, use activity bouts algorithm from the NCI's SAS programs [1]; if FALSE, use regular algorithm (see Details).
- **sed.bout.tol**: Number of minutes with counts outside sedentary range allowed during sedentary bouts.
- **sed.bout.tol.maximum**: Upper cut-off for count values outside sedentary range during a sedentary bout.
- **artifact.axis**: Axis that should be used to detect artifacts. Should be one of "vert", "ap", "ml", "sum" (for triaxial sum), or "mag" (for triaxial vector magnitude).
- **artifact.thresh**: Lower cut-off for counts that are abnormally high and should be considered artifacts.
- **artifact.action**: If 1, exclude days that have one or more artifacts; if 2, consider artifacts as non-wear time; if 3, replace artifacts with average of neighboring count values (for all axes of minutes identified as artifacts); if 4, take no action.
- **weekday.weekend**: If TRUE, function computes physical activity averages for weekdays and weekend days separately (in addition to daily averages for all valid days, which are computed regardless). If FALSE, function only computes averages for all valid days.
- **return.form**: If 1, function returns physical activity variables on per-person basis, i.e. daily averages for each participant; if 2, function returns variables on per-day basis; if 3, function returns both via a list.

**Details**

The algorithm used to identify non-wear time is defined by function inputs nonwear.axis, nonwear.window, nonwear.tol, nonwear.tol.upper, and nonwear.nci. If nonwear.nci is set to FALSE, a 'regular' non-wear algorithm is used. This algorithm classifies as non-wear time any interval
of length nonwear.window in which no more than nonwear.tol counts in the nonwear.axis axis are non-zero, and those counts are all less than nonwear.tol.upper. If nonwear.nci is set to TRUE, the non-wear algorithm from the NCI’s SAS programs [2] is used. This algorithm classifies as non-wear time any interval of length nonwear.window that starts with a count value of 0, does not contain any periods with (nonwear.tol + 1) consecutive non-zero count values, and does not contain any counts greater than nonwear.tol.upper. Once a non-wear bout is established, it continues until there are (nonwear.tol + 1) consecutive non-zero count values or a single count value greater than nonwear.tol.upper.

The activity bout algorithm operates similarly to the non-wear algorithm. If active.bout.nci is set to FALSE, a ‘regular’ algorithm is used. To illustrate, any interval of length active.bout.length where no more than active.bout.tol minutes have counts less than int.cuts[3], and the counts below int.cuts[3] are all mvpa.bout.tol.lower or greater, is considered an MVPA bout. If active.bout.nci is set to TRUE, the NCI’s algorithm is used. This algorithm defines an MVPA bout as an interval that starts with ten consecutive count values greater than or equal to int.cuts[3], allowing for up to active.bout.tol minutes with counts below int.cuts[3]. The first minute of the bout cannot be below int.cuts[3]. Once the MVPA bout is established, it continues until there are (active.bout.tol + 1) consecutive minutes with counts less than int.cuts[3]. The parameters mvpa.bout.tol.lower and vig.bout.tol.lower are not used in the NCI bout algorithm.

Value

A single matrix or a list of two matrices, depending on return.form.

Note

This function is similar to accel.process.uni, but works with triaxial accelerometer data. For example, rather than just calculating a single counts variable for each day of monitoring, accel.process.tri calculates a daily counts variable for the vertical axis, anteroposterior (AP) axis, and mediolateral (ML) axis, and also for the triaxial sum and triaxial vector magnitude (defined as the square root of the sum of the squares of the three axes). Also, the user can choose which of the five signals to use for artifact correction, non-wear detection, activity intensity classification, and hourly counts per minute averaging.

Some additional information on the package accelerometry and its functions can be found on the author’s website, https://sites.google.com/site/danevandomelen/.

Author(s)

Dane R. Van Domelen

References


Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.
See Also

accel.artifacts, accel.bouts, accel.process.uni, accel.intensities, accel.sedbreaks,
accel.weartime, blockaves, movingaves, rle, inverse.rle

Examples

# Load in sample matrix
data(tridata)

# Process data and request per-day variables
accel.days1 <- accel.process.uni(counts = tridata[, 1:3], steps = tridata[, 4])

# Process data, but for non-wear detection use triaxial vector magnitude with 90-
# minute window and two-minute tolerance for nonzero counts up to 200
accel.days2 <- accel.process.uni(counts = tridata[, 1:3], steps = tridata[, 4],
nonwear.axis = "mag", nonwear.window = 90,
nonwear.tol = 2, nonwear.tol.upper = 200)

accel.process.uni  Process Uniaxial Minute-to-Minute Accelerometer Data

Description

This function calculates a variety of physical activity variables based on uniaxial minute-to-minute
accelerometer data for individual participants. A data dictionary for the variables returned is available here: https://sites.google.com/site/danevandomelemen/r-package-accelerometry/data-dictionary.

Usage

accel.process.uni(counts, steps = NULL, nci.methods = FALSE,
start.date = as.Date("2014/1/5"), start.time = "00:00:00", id = NULL,
brevity = 1, valid.days = 1, valid.week.days = 0,
valid.weekend.days = 0, int.cuts = c(100, 760, 2020, 5999),
cpm.nci = FALSE, days.distinct = FALSE, nonwear.window = 60,
nonwear.tol = 0, nonwear.tol.upper = 99, nonwear.nci = FALSE,
weartime.min = 600, weartime.max = 1440,
partialday.min = 1440, active.bout.len = 10,
active.bout.tol = 0, mvp.bout.tol.lower = 0, vig.bout.tol.lower = 0,
active.bout.nci = FALSE, sed.bout.tol = 0,
sed.bout.tol.max = int.cuts[2] - 1, artifact.thresh = 25000,
artifact.action = 1, weekday.weekend = FALSE, return.form = 2)

Arguments

counts  Time series accelerometer counts vector
steps   Steps vector. If specified, must be same length as counts.
nci.methods  If TRUE, inputs are set to replicate the data processing methods used by the
NCT’s SAS programs [2]. More specifically:
valid.days = 4; valid.week.days = 0; valid.weekend.days = 0; int.cuts = c(100, 760, 2020, 5999); cpm.nci = TRUE; days.distinct = TRUE; nonwear.window = 60; nonwear.tol = 2; nonwear.tol.upper = 100; nonwear.nci = TRUE; weartime.minimum = 600; weartime.maximum = 1440; partialday.minimum = 1440; active.bout.length = 10; active.bout.tol = 2; mvpa.bout.tol.lower = 0; vig.bout.tol.lower = 0; active.bout.nci = TRUE; sed.bout.tol = 0; sed.bout.tol.maximum = 759; artifact.thresh = 32767; artifact.action = 3
Even if nci.methods is set to TRUE, the user can specify non-default values for
brevity, weekday.weekend, and return.form.

start.date  Date of first day of monitoring (must be of class ‘date’). Only used to extract
day of week, so if day of week is known but date is not, user can enter any date
that corresponds to that day of the week. The default date corresponds to the
first Sunday in January 2014.

start.time  Optional character vector indicating the start time for monitoring. If not speci-
ﬁed it is assumed to be 00:00:00, i.e. the very beginning of the ﬁrst day.

id  Either a single value or a vector indicating the ID number for the participant
whose accelerometer data was entered.

brevity  Controls the number of physical activity variables returned. If 1, returns basic
indicators of physical activity volume; if 2, also returns indicators of activity
intensities, activity bouts, sedentary behavior, and peak activity; if 3, also returns
hourly count averages.

valid.days  Minimum number of valid days to be considered valid for analysis.
valid.week.days  Minimum number of valid weekdays to be considered valid for analysis.
valid.weekend.days  Minimum number of valid weekend days to be considered valid for analysis.
int.cuts  Vector of four cut-points from which ﬁve intensity ranges are derived. For example,
if thresh = c(100, 760, 2020, 5999), minutes with 0-99 counts are classiﬁed
as intensity level 1, minutes with 100-759 counts are classiﬁed as intensity level
2, ... , and minutes with 5999 or greater counts are classiﬁed as intensity level 5.
Intensities 1-5 typically correspond to sedentary, light, lifestyle, moderate, and
vigorous.

cpm.nci  If TRUE, average counts per minute is calculated by dividing average daily
counts by average daily weartime, as opposed to averaging each day’s counts
per minute value. In general, leave as FALSE unless you want to replicate the
NCT’s SAS programs [2].

days.distinct  If TRUE, treat each day of data as distinct, i.e. identify non-wear time and ac-
tivity bouts in day 1, then day 2, etc.; If FALSE, apply algorithms on continuous
basis for full monitoring period. If protocol has participants remove accelerome-
ter for sleep, strongly recommend setting to FALSE to capture non-wear periods
that start between 11 pm and midnight.

nonwear.window  Minimum length of a non-wear interval.
nonwear.tol  Number of minutes with non-zero counts allowed during a non-wear interval.
nonwear.tol.upper
Maximum count value for a minute with non-zero counts during a non-wear interval.

nonwear.nci
If TRUE, use non-wear algorithm from the NCI’s SAS programs [2]; if FALSE, use regular algorithm (see Details).

weartime.minimum
Minimum number of wear time minutes for a day of monitoring to be considered valid.

weartime.maximum
Maximum number of wear time minutes for a day of monitoring to be considered valid. The default is 1440, but it may be better to set it to a lower value (e.g. 1200) if participants were instructed to remove their accelerometers for sleeping. Daily wear time greater than 1200 minutes corresponds to less than 4 hours of sleep. In these cases it seems more likely that the participant slept while wearing the device, and as a result had small movements overnight show up as wear time. This could inflate estimates of sedentary time and shrink estimates of physical activity, e.g. counts per minute.

partialday.minimum
Minimum number of minutes for a partial day of monitoring to be processed and potentially considered valid for analysis (generally applies only to the first and last days of monitoring, which may not cover full 24-hour periods). This input is included because some researchers may prefer to exclude a day that only has data from, say, 1 pm to midnight. Even though there may be sufficient wear time during that period to be classified as a valid day, the missing chunk of data prior to 1 pm may result in the day not being representative of the participant’s usual physical activity.

active.bout.length
Minimum length of moderate-to-vigorous physical activity (MVPA) and vigorous physical activity (VPA) bouts.

active.bout.tol
Number of minutes with counts below the required intensity level allowed during MVPA and VPA bouts. If set to a non-zero value, and active.bout.nci is FALSE, specifying non-zero values for mvpa.bout.tol.lower and vig.bout.tol.lower is highly recommended. Otherwise the algorithm will tend to classify minutes immediately before and after an activity bout as being part of the bout.

mvpa.bout.tol.lower
Lower cut-off for count values outside of MVPA intensity range during an MVPA bout.

vig.bout.tol.lower
Lower cut-off for count values outside of VPA intensity range during a VPA bout.

active.bout.nci
If TRUE, use activity bouts algorithm from the NCI’s SAS programs [2]; if FALSE, use regular algorithm (see Details).

sed.bout.tol
Number of minutes with counts outside sedentary range allowed during sedentary bouts.

sed.bout.tol.maximum
Upper cut-off for count values outside sedentary range during a sedentary bout.
artifact.thresh
Lower cut-off for counts that are abnormally high and should be considered
artifacts (see Note).

artifact.action
If 1, exclude days that have one or more artifacts; if 2, consider artifacts as non-
wear time; if 3, replace artifacts with average of neighboring count values; if 4,
take no action (see Note).

weekday.weekend
If TRUE, function computes physical activity averages for weekdays and week-
end days separately (in addition to daily averages for all valid days, which are
computed regardless). If FALSE, function only computes averages for all valid
days.

return.form
If 1, function returns physical activity variables on per-person basis, i.e. daily
averages for each participant; if 2, function returns variables on per-day basis; if
3, function returns both via a list.

Details
The algorithm used to identify non-wear time is defined by function inputs nonwear.window, non-
wear.tol, nonwear.tol.upper, and nonwear.nci. If nonwear.nci is set to FALSE, a ‘regular’ non-wear
algorithm is used. This algorithm classifies as non-wear time any interval of length nonwear.window
in which no more than nonwear.tol counts are non-zero, and those counts are all less than non-
wear.tol.upper. If nonwear.nci is set to TRUE, the non-wear algorithm from the NCI’s SAS pro-
grams [2] is used. This algorithm classifies as non-wear time any interval of length nonwear.window
that starts with a count value of 0, does not contain any periods with (nonwear.tol + 1) consecutive
non-zero count values, and does not contain any counts greater than nonwear.tol.upper. Once a non-
wear bout is established, it continues until there are (nonwear.tol + 1) consecutive non-zero count
values or a single count value greater than nonwear.tol.upper.

The activity bout algorithm operates similarly to the non-wear algorithm. If active.bout.nci is set
to FALSE, a ‘regular’ algorithm is used. To illustrate, any interval of length active.bout.length
where no more than active.bout.tol minutes have counts less than int.cuts[3], and the counts below
int.cuts[3] are all mvpa.bout.tol.lower or greater, is considered an MVPA bout. If active.bout.nci
is set to TRUE, the NCI’s algorithm is used. This algorithm defines an MVPA bout as an interval
that starts with ten consecutive count values greater than or equal to int.cuts[3], allowing for up to
active.bout.tol minutes with counts below int.cuts[3]. The first minute of the bout cannot be
below int.cuts[3]. Once the MVPA bout is established, it continues until there are (active.bout.tol +
1) consecutive minutes with counts less than int.cuts[3]. The parameters mvpa.bout.tol.lower and
vig.bout.tol.lower are not used in the NCI bout algorithm.

Value
A single matrix or a list of two matrices, depending on return.form.

Note
There is no perfect solution for dealing with abnormally high count values, also known as arti-
facts. The NCI’s SAS programs replace artifacts (which they define as the ActiGraph AM-7164
maximum, 32767) with the mean of neighboring count values [2]. This can be done by specifying
artifact.thresh = 32767 and artifact.action = 3. This method may work well, but in many cases count values that are artifacts are surrounded by count values that are only slightly lower, bringing into question whether the entire group of counts is plausible or implausible. Count values at or around the cut-point defined by artifact.thresh can contribute greatly to daily counts. Therefore the default settings, artifact.thresh = 25000 and artifact.action = 1, simply excludes days of monitoring with one or more count values of 25000 or greater. As this solution is clearly not ideal, users are welcome to choose their own preferred setting for artifact.thresh and artifact.action.

Some additional information on the package accelerometry and its functions can be found on the author’s website, https://sites.google.com/site/danevandomelen/.

Author(s)
Dane R. Van Domelen

References


Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.

See Also
accel.artifacts, accel.bouts, accel.process.tri, accel.intensities, accel.sedbreaks, accel.weartime, blockaves, movingaves, rleR, inverseRleR

Examples

# Load in sample matrix
data(unidata)

# Get data from ID number 21005
id.part1 <- unidata[unidata[, "seqn"] == 21005, "seqn"]
counts.part1 <- unidata[unidata[, "seqn"] == 21005, "paxinten"]

# Process data from ID 21005 and request per-day variables
accel.days <- accel.process.uni(counts = counts.part1, id = id.part1)

# Process data from ID 21005 and request daily averages
accel.averages <- accel.process.uni(counts = counts.part1, id = id.part1,
                                 return.form = 1)

# Process data from ID 21005 and request per-day variables and daily averages
accel.list <- accel.process.uni(counts = counts.part1, id = id.part1,
                                return.form = 3)
# Process data according to methods used in NCI's SAS programs [2]
accel.nci1 <- accel.process.uni(counts = counts.part1, id = id.part1, brevity = 2,
valid.days = 4, cpm.nci = TRUE, days.distinct = TRUE,
nonwear.tol = 2, nonwear.tol.upper = 100,
nonwear.nci = TRUE, weartime.maximum = 1440,
active.bout.tol = 2, active.bout.nci = TRUE,
artifact.thresh = 32767, artifact.action = 3,
return.form = 3)

# Repeat, but use nci.methods input for convenience
accel.nci2 <- accel.process.uni(counts = counts.part1, id = id.part1, nci.methods = TRUE,
brevity = 2, return.form = 3)

# Verify that previous two function calls give identical results
all(accel.nci1[[1]] == accel.nci2[[1]])
all(accel.nci1[[2]] == accel.nci2[[2]])

---

accel.sedbreaks  
\textit{Sedentary Breaks Detection}

\section*{Description}
This function identifies sedentary breaks in minute-to-minute accelerometer data.

\section*{Usage}
accel.sedbreaks(counts, weartime = NULL, thresh = 100, return.flags = FALSE,
skipchecks = FALSE)

\section*{Arguments}

\begin{itemize}
\item \texttt{counts} \hspace{1cm} Time series accelerometer counts vector.
\item \texttt{weartime} \hspace{1cm} Accelerometer wear time vector; must be same length as counts and consist of 1's and 0's (if specified).
\item \texttt{thresh} \hspace{1cm} Minimum count value that must be achieved to record a sedentary break.
\item \texttt{return.flags} \hspace{1cm} If TRUE, function returns vector of 0's and 1's in which 1's indicate minutes in which a sedentary break has occurred; if FALSE, function returns total number of sedentary breaks during monitoring period.
\item \texttt{skipchecks} \hspace{1cm} If TRUE, function skips error checking code and runs slightly faster.
\end{itemize}

\section*{Value}
Either a single numeric value indicating the total number of sedentary breaks during the monitoring period, or a vector of same length as counts where 1's indicate minutes in which a sedentary break has occurred.
Note

If a wear time vector is not provided, algorithm may consider the first minute after a non-wear period as a sedentary break, which is not correct because the preceding minute was non-wear time and may or may not have been sedentary time. This could result in a slight over-estimate of sedentary breaks.

Some additional information on the package accelerometry and its functions can be found on the author's website, https://sites.google.com/site/danevandomelen/

Author(s)

Dane R. Van Domelen

References

Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.

See Also

accel.weartime, accel.process.uni, accel.process.tri

Examples

# Load in sample data frame
data(unidata)

# Get data from ID number 21005
id.part1 <- unidata[unidata[, "seqn"] == 21005, "seqn"]
counts.part1 <- unidata[unidata[, "seqn"] == 21005, "paxinten"]

# Identify periods of valid wear time
weartime.flag <- accel.weartime(counts = counts.part1)

# Count number of sedentary breaks (over full week)
um.sedbreaks <- accel.sedbreaks(counts = counts.part1, weartime = weartime.flag)

# Flag minutes that represent sedentary breaks
flag.sedbreaks <- accel.sedbreaks(counts = counts.part1, weartime = weartime.flag, return.flags = TRUE)

accel.weartime Accelerometer Non-Wear Detection

Description

This function identifies periods of non-wear time in minute-to-minute accelerometer data.
Usage

```r
accel.weartime(counts, window = 60, tol = 0, tol.upper = 99, nci = FALSE,
                 days.distinct = FALSE, skipchecks = FALSE)
```

Arguments

- **counts**: Time series accelerometer counts vector.
- **window**: Minimum length of a non-wear interval.
- **tol**: Number of minutes with non-zero counts allowed during a non-wear interval.
- **tol.upper**: Maximum count value for a minute with non-zero counts during a non-wear interval.
- **nci**: If TRUE, use non-wear algorithm from the NCI’s SAS programs [1]; if FALSE, use regular algorithm (see Details).
- **days.distinct**: If TRUE, treat each day of data as distinct, i.e. identify wear time in day 1, then day 2, etc.; if FALSE, apply algorithm on continuous basis for full monitoring period. Function assumes that first 1440 data points are day 1, next 1440 are day 2, and so on.
- **skipchecks**: If TRUE, function skips error checking code and runs slightly faster.

Details

If `nci` is set to FALSE, the algorithm uses a moving window to go through every possible interval of length `window` in input vector `counts`. Any interval in which no more than `tol` counts are non-zero, and those counts are less than `tol.upper`, is classified as non-wear time.

If `nci` is set to TRUE, non-wear time is classified according to the algorithm used in the NCI’s SAS programs [1]. Briefly, this algorithm defines a non-wear period as an interval of length `window` that starts with a count value of 0, does not contain any periods with `(tol + 1)` consecutive non-zero count values, and does not contain any counts greater than `tol.upper`. If these criteria are met, the bout continues until there are `(tol + 1)` consecutive non-zero count values or a single count value greater than `tol.upper`.

Value

Integer vector of same length as `counts` and `weartime`, with 1’s indicating minutes of valid wear time, and 0’s indicating minutes of non-wear time.

Note

Leaving `days.distinct` set to FALSE is strongly encouraged. If set to TRUE, non-wear periods that span the end of one day and the beginning of the next might be missed. This option is included because it is necessary to reproduce the algorithm used in the NCI’s SAS programs [1].

Some additional information on the package accelerometry and its functions can be found on the author’s website, https://sites.google.com/site/danevandomelen/

Author(s)

Dane R. Van Domelen
References


Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.

See Also

accel.process.uni, accel.process.tri

Examples

# Load in sample data frame
data(unidata)

# Get data from ID number 21005
counts.part1 <- unidata[unidata[, "seqn"] == 21005, "paxinten"]

# Identify periods of valid wear time
weartime.flag <- accel.weartime(counts = counts.part1)

blockaves

Block Average Calculator

Description

This function returns averages for non-overlapping segments of data.

Usage

blockaves(x, window, skipchecks = FALSE)

Arguments

x Input vector.
window Window length.
skipchecks If TRUE, function skips error checking code and runs slightly faster.

Value

Vector of length floor(length(x) / window)
Note

If length(x) is not an exact multiple of window, the average for the last partial segment of data is dropped.

Some additional information on the package accelerometry and its functions can be found on the author’s website, https://sites.google.com/site/danevandomelen/

Author(s)

Dane R. Van Domelen

References

Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.

See Also

movingaves

Examples

# Load in sample data frame
data(unidata)

# Get data from ID number 21005, Saturday only
counts.saturday <- unidata[unidata[, "seqn"] == 21005 & unidata[, "paxday"] == 7, "paxinten"]

# Calculate and plot hourly count averages
hourly.averages <- blockaves(x = counts.saturday, window = 60)
plot(hourly.averages)

---

inverse.rle2

Inverse Run Length Encoding (Alternate Implementation)

Description

Re-construct vector compressed by rle2.

Usage

inverse.rle2(x)

Arguments

x

Object returned by rle2.
Value

Numeric or character vector.

Note

This function expands a vector compressed by rle2. It basically just re-formats the object generated by rle2 and then calls inverse.rle. It also works on vectors compressed by rle, but there is no advantage to using inverse.rle2 rather than inverse.rle in that scenario.

Author(s)

Dane R. Van Domelen

References

Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.

See Also

rle2, rle, inverse.rle

Examples

# Create dummy vector x
x <- c(0, 0, 0, -1, -1, 10, 10, 4, 6, 6)

# Summarize x using rle2
x.summarized <- rle2(x)

# Reconstruct x
x.reconstructed <- inverse.rle2(x.summarized)


description

This function returns either a vector of moving averages or the maximum moving average for some input vector x.

Usage

movingaves(x, window, return.max = FALSE, skipchecks = FALSE)
movingaves

Arguments

- `x`: Input vector.
- `window`: Window length for moving averages.
- `return.max`: If TRUE, function returns the maximum moving average; if FALSE, function returns a vector of moving averages.
- `skipchecks`: If TRUE, function skips error checking code and runs slightly faster.

Value

Either a single numeric value indicating the maximum moving average, or a vector of length \((\text{length}(x) - \text{window} + 1)\) with moving averages.

Note

Some additional information on the package accelerometry and its functions can be found on the author’s website, https://sites.google.com/site/danevandomelen/

Author(s)

Dane R. Van Domelen

References

Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.

See Also

- `blockaves`

Examples

```r
# Load in sample data frame
data(unidata)

# Get data from ID number 21005
counts.part1 <- unidata[, seqn == 21005, paxinten]

# Create vector of all 10-minute moving averages
all.10min.averages <- movingaves(x = counts.part1, window = 10)

# Calculate maximum 10-minute moving average
max.10min.average <- movingaves(x = counts.part1, window = 10, return.max = TRUE)
```
Function for Calculating Daily Averages for Physical Activity Variables

Description

Not intended for direct use.

Usage

`personvars(dayvars, rows, days, wk, we)`

Arguments

- `dayvars` Matrix that contains daily physical activity variables.
- `rows` Number of rows in the matrix to be returned.
- `days` Minimum number of valid days required to be considered valid data for analysis.
- `wk` Minimum number of valid weekdays required to be considered valid data for analysis.
- `we` Minimum number of valid weekend days required to be considered valid data for analysis.

Value

Matrix with averages for physical activity variables.

Author(s)

Dane R. Van Domelen

References

Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.
rle2  

*Run Length Encoding (Alternate Implementation)*

**Description**

Summarize vector of numeric or character values containing runs of repeated values. This function is very similar to the base function rle, but sometimes much faster, and different in that it has an option to return the start/end indices for each run.

**Usage**

```r
rle2(x, indices = FALSE, return.list = FALSE)
```

**Arguments**

- `x`  
  Input vector containing either numeric or character data.
- `indices`  
  If FALSE, function records values and lengths for each run; if TRUE, function records values, start positions, stop positions, and lengths for each run.
- `return.list`  
  If FALSE, function returns 2- or 4-column matrix if `x` is a numeric vector and 2- or 4- column data frame if `x` is a character vector (number of columns depends on value for `indices`); if TRUE, function returns 2- or 4-element list, similar to the object returned by the base R function rle.

**Value**

Depending on the inputs `indices` and `return.list`, a matrix or data frame with 2 or 4 columns, or a list of either 2 or 4 vectors.

**Note**

For numeric data, rle2 runs 2-10 times faster than rle. In general, the longer the input vector and the longer the runs, the greater the speed advantage of rle2 over rle.

For character data, rle2 is often slower than rle, sometimes by an order of magnitude or more. However, for very long vectors (e.g. length > 10,000) with long runs (e.g. average run length > 100), rle2 can be up to 5 times faster than rle.

Some additional information on the package accelerometry and its functions can be found on the author’s website, [https://sites.google.com/site/danevandomelen/](https://sites.google.com/site/danevandomelen/)

**Author(s)**

Dane R. Van Domelen

**References**

Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.
See Also

inverse.rle2, rle, inverse.rle

Examples

# Create dummy vector x
x <- c(0, 0, 0, -1, -1, 10, 10, 1, 4, 6, 6)

# Summarize x using rle2
x_summary <- rle2(x)

# Repeat, but also record start/stop indices for each run
x_summary <- rle2(x = x, indices = TRUE)

---

tridata

Triaxial Sample Data

description

This dataset is used to illustrate the accel.process.tri function in the accelerometry package.

Usage

data(tridata)

Format

The format is: int [1:10080, 1:4] 118 0 0 2 0 0 2 0 0 2 0 0 ... - attr(*, "dimnames")=List of 2 ..$ : chr [1:10080] "526" "527" "528" "529" ... ..$ : chr [1:4] "vert" "ap" "ml" "steps"

Details

This dataset contains a four-column matrix. The first three columns are counts in the vertical, anteroposterior, and mediolateral axes, and the fourth column is steps. There are seven days worth of data from a volunteer.

Source

This data was graciously provided by Ei Ei Khaing Nang from the Department of Epidemiology and Public Health, Yong Loo Lin School of Medicine, National University of Singapore, Singapore, Republic of Singapore.

References

Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.
Examples

# Load in sample data
data(uniData)

# Plot one day of step data
plot(uniData[1:1440, 4])

---

**uniData**  
*Uniaxial Sample Data*

Description

This dataset is used to illustrate the various functions in the package `accelerometry`.

Usage

```r
data(unidata)
```

Format

The format is: `int [1:50400, 1:3] 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 21005 ...` - attr(*, "dimnames")=List of 2  
`. `$ : NULL .` $ : chr [1:3] "seqn" "paxday" "paxinten"

Details

NA

Source

The data is for from five participants in NHANES 2003-2004 [1]. The full dataset is available at:  

References


Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.
Examples

# Load in sample data
data(unidata)

# Plot full week of data from ID number 21007
counts.part3 <- unidata[unidata[, "seqn"] == 21007, "paxinten"]
plot(counts.part3)
Index

*Topic accelerometry
   accel.artifacts, 4
   accel.bouts, 5
   accel.intensities, 7
   accel.process.tri, 8
   accel.process.uni, 13
   accel.sedbreaks, 18
   accel.weartime, 19
   blockaves, 21
   movingaves, 23
   personvars, 25
   rle2, 26
*Topic artifacts
   accel.artifacts, 4
*Topic block average
   blockaves, 21
*Topic bouts
   accel.bouts, 5
*Topic consecutive
   rle2, 26
*Topic datasets
   tridata, 27
   unidata, 28
*Topic intensity
   accel.intensities, 7
*Topic moving average
   movingaves, 23
*Topic non-wear time
   accel.weartime, 19
*Topic package
   accelerometry-package, 2
*Topic physical activity
   accel.process.tri, 8
   accel.process.uni, 13
*Topic rle
   inverse.rle2, 22
   rle2, 26
*Topic sedentary breaks
   accel.sedbreaks, 18

*Topic triaxial
   accel.process.tri, 8
*Topic wear time
   accel.weartime, 19
   accel.artifacts, 2, 4, 13, 17
   accel.bouts, 2, 5, 13, 17
   accel.intensities, 2, 7, 13, 17
   accel.process.tri, 2, 5, 7, 8, 17, 19, 21
   accel.process.uni, 2, 5, 7, 8, 12, 13, 13, 19, 21
   accel.sedbreaks, 2, 13, 17, 18
   accel.weartime, 2, 7, 8, 13, 17, 19, 19
   accelerometry (accelerometry-package), 2
   accelerometry-package, 2
   blockaves, 2, 13, 17, 21, 24
   inverse.rle, 23, 27
   inverse.rle2, 2, 13, 17, 22, 27
   movingaves, 2, 13, 17, 22, 23
   personvars, 25
   rle, 23, 27
   rle2, 2, 13, 17, 23, 26
   tridata, 27
   unidata, 28