

# Package ‘activityCounts’

July 31, 2019

**Type** Package

**Title** Generate ActiLife Counts

**Version** 0.1.2

**Description** ActiLife software generates activity counts from data collected by Actigraph accelerometers <[https://s3.amazonaws.com/actigraphcorp.com/wp-content/uploads/2017/11/26205758/ActiGraph-White-Paper\\_What-is-a-Count\\_.pdf](https://s3.amazonaws.com/actigraphcorp.com/wp-content/uploads/2017/11/26205758/ActiGraph-White-Paper_What-is-a-Count_.pdf)>.

Actigraph is one of the most common research-grade accelerometers. There is considerable research validating and developing algorithms for human activity using ActiLife counts. Unfortunately, ActiLife counts are proprietary and difficult to implement if researchers use different accelerometer brands.

The code creates ActiLife counts from raw acceleration data for different accelerometer brands and it is developed

based on the study done by Brond and others (2017) <[doi:10.1249/MSS.0000000000001344](https://doi.org/10.1249/MSS.0000000000001344)>.

**URL** <https://github.com/walkabillylab/activityCounts>,  
<https://github.com/jbrond/ActigraphCounts>

**BugReports** <https://github.com/walkabillylab/activityCounts/issues>

**Depends** R (>= 2.10)

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.1.1

**Suggests** knitr, rmarkdown, ggplot2

**VignetteBuilder** knitr

**Imports** seewave, signal, tibble, lubridate, magrittr

**NeedsCompilation** no

**Author** Ruben Brondeel [aut],  
Javad Rahimipour Anaraki [aut],  
SeyedJavad KhataeiPour [aut, cre],  
Daniel Fuller [aut, cph],  
Beap Lab [cph]

**Maintainer** SeyedJavad KhataeiPour <skhataeipour@mun.ca>

**Repository** CRAN

**Date/Publication** 2019-07-31 07:30:04 UTC

## R topics documented:

counts	2
pptrunc	3
runsum	4
sampleCounts	4
sampleXYZ	5
trunc	5

---

counts

*counts*

---

### Description

Calculates ActiLife counts based on raw accelerometer data

### Usage

```
counts(data, hertz = -1, x_axis = 2, y_axis = 3, z_axis = 4,
       time_column = -1, start_time = -1)
```

### Arguments

data	Accelerometer data, Must have at least three columns.
hertz	Sampling frequency in Hz
x_axis	Indicates the column number which has the accel data for x direction, the default is 2
y_axis	Indicates the column number which has the accel data for y direction, the default is 3
z_axis	Indicates the column number which has the accel data for z direction, the default is 4
time_column	Optional. Indicates the column number which has the date and time. The first row will be considered as the start time of the study. You can use the "start_time" argument to provide the start time explicitly.
start_time	Optional. Use this to define the start time of the experiment. You can use this argument if the data does not contain a time column.

**Value**

Returns a `data.table` with four columns:

**Time** The start time of the measurement

**x** the number of counts for X axis

**y** the number of counts for Y axis

**z** the number of counts for Z axis

**See Also**

`sampleXYZ` raw accelerometer data for testing `counts()` function.

`sampleCounts` counts calculated by `activityCounts` and `ActiLife`

**Examples**

```
# for the sampleXYZ dataset, sampling frequency is 100 Hz
counts(data = sampleXYZ, hertz = 100)

# when start time is given explicitly
study_start_time <- "2017-08-22 12:30:10"
counts(data = sampleXYZ, hertz = 100 , start_time = study_start_time)

# the data has a time column, which is the first column
counts(data = sampleXYZ, hertz = 100 , time_column = 1)

# explicitly specify the X, Y and Z axis columns.
counts(data = sampleXYZ, hertz = 100 , x_axis = 2, y_axis = 3, z_axis = 4)
```

---

pptrunc

*pptrunc*

---

**Description**

pptrunc

**Usage**

```
pptrunc(data, max_value)
```

**Arguments**

`data` The variable that will be truncated

`max_value` The upper bound ( `-max_value` is the lower bound)

**Value**

the highest(or the lowest) value of "data" and "max\_value"

---

runsum	<i>runsum</i>
--------	---------------

---

**Description**

runsum

**Usage**

```
runsum(data, len, threshold)
```

**Arguments**

data	input data
len	the length
threshold	the threshold

**Value**

returns a

---

sampleCounts	<i>The counts calculated by activityCounts and ActiLife based on included raw accelerometer data</i>
--------------	--

---

**Description**

A simple `data.table` which its first row is measurement time. Then for each time step, counts are calculated by `activityCounts` and the ActiLife software. The counts are calculated based on included `sampleXYZ` dataset, which its sampling frequency is 100H.

**Usage**

```
sampleCounts
```

**Format**

A `data.table` with nine columns:

**Time** Date and time

**activityCounts\_x\_counts** counts calculated by `counts()` function in X direction

**activityCounts\_y\_counts** counts calculated by `counts()` function in Y direction

**activityCounts\_z\_counts** counts calculated by `counts()` function in Z direction

**ActiLife\_x\_counts** counts calculated by ActiLife software in X direction

**ActiLife\_y\_counts** counts calculated by ActiLife software in Y direction

**ActiLife\_z\_counts** counts calculated by ActiLife software in Z direction

**See Also**

`counts` to see how to produce counts.

`sampleXYZ` raw accelerometer data for testing `counts()` function.

---

sampleXYZ

*Raw accelerometer data for the activityCounts package*

---

**Description**

A simple `data.table` that contains raw accelerometer data for testing the `counts` function. Sampling frequency of this `data.table` is 100Hz, therefore pass 100 as the second argument when using the `counts` function.

**Usage**

```
sampleXYZ
```

**Format**

A `data.table` with four columns:

**Time** Timestamp

**accelerometer\_X** accelerometer data in X direction

**accelerometer\_Y** accelerometer data in Y direction

**accelerometer\_Z** accelerometer data in Z direction

**See Also**

`counts` to see how to produce counts.

`sampleCounts` counts calculated by `activityCounts` and `ActiLife`

---

trunc

*trunc*

---

**Description**

trunc

**Usage**

```
trunc(data, min_value)
```

**Arguments**

data	The input variable which will be altered if less than the threshold
min_value	the threshold which the input below it will be set to zero

**Value**

returns zero if the "data" is less than the "mean\_value" otherwise returns the "data"