Package ‘MIMSunit’

August 1, 2020

Type Package

Title Algorithm to Compute Monitor Independent Movement Summary Unit (MIMS-Unit)

Version 0.9.2

Date 2020-07-30

Description The MIMS-unit algorithm is developed to compute Monitor Independent Movement Summary Unit, a measurement to summarize raw accelerometer data while ensuring harmonized results across different devices. It also includes scripts to reproduce results in the related publication (John, D., Tang. Q., Albinali, F. and Intille, S. (2019) <doi:10.1123/jmpb.2018-0068>).

License MIT + file LICENSE

Language en-US

Depends R (>= 3.6.0)

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

URL https://mhealthgroup.github.io/MIMSunit/, https://github.com/mhealthgroup/MIMSunit/tree/master

BugReports https://github.com/mhealthgroup/MIMSunit/issues/

Imports caTools (>= 1.17.1.1), dplyr (>= 0.7.7), lubridate (>= 1.7.4), magrittr (>= 1.5), matlab (>= 1.0.2), plyr (>= 1.8.4), readr (>= 1.1.1), R.utils (>= 2.7.0), signal (>= 0.7-6), stringr (>= 1.3.1), xts (>= 0.11-2), dygraphs (>= 1.1.1.6), shiny (>= 1.4.0.2), RColorBrewer (>= 1.1-2), utils (>= 3.6.1), ggplot2 (>= 3.2.1)

Suggests testthat, pkgdown, gridExtra, remotes

SystemRequirements memory (>= 4GB) Ubuntu: build-essential, libxml2-dev, libssl-dev, libcurl4-openssl-dev Windows: Rtools (>= 3.5)

NeedsCompilation no

Author Qu Tang [aut, cre] (<https://orcid.org/0000-0001-5415-0205>),
Dinesh John [aut],
Stephen Intille [aut],
mHealth Research Group [cph] (https://mhealthgroup.org)

Maintainer Qu Tang <tang.q@northeastern.edu>

Repository CRAN

Date/Publication 2020-08-01 00:20:10 UTC

R topics documented:

`aggregate_for_mims` *Aggregate over epoch to get numerically integrated values.*

Description

`aggregate_for_mims` returns a dataframe with integrated values by trapzoidal method over each epoch for each column. The epoch start time will be used as timestamp in the first column.

Usage

```
aggregate_for_mims(df, epoch, method = "trapz", rectify = TRUE, st = NULL)
```

Arguments

`df`	dataframe of accelerometer data in mhealth format. First column should be timestamps in POSIXt format.
`epoch`	string. Any format that is acceptable by argument `breaks` in method `cut.POSIXt`.For example, "1 sec", "1 min", "5 secs", "10 mins".
`method`	string. Integration methods. Supported strings include: "trapz", "power", "sum", "meanBySecond", "meanBySize". Default is "trapz".
`rectify`	logical. If TRUE, input data will be rectified before integration. Default is TRUE.
`st`	character or POSIXct timestamp. An optional start time you can set to force the epochs generated by referencing this start time. If it is NULL, the function will use the first timestamp in the timestamp column as start time to generate epochs. This is useful when you are processing a stream of data and want to use a common start time for segmenting data. Default is NULL.

Details

This function accepts a dataframe (in mhealth accelerometer data format) and computes its aggregated values over each fixed epoch using different integration methods (default is trapzoidal method, other methods are not used by mims unit algorithm) for each value columns. The returned dataframe will have the same number of columns as input dataframe, and have the same datetime format as input dataframe in the timestamp column. The trapzoidal method used in the function is based on `trapz`.

Value

dataframe. The returned dataframe will have the same format as input dataframe.

How is it used in mims-unit algorithm?

This function is used in mims-unit algorithm after filtering (`iir`). The filtered signal will be rectified and integrated to get mims unit values for each axis using this function.

Note

If epoch argument is not provided or is `NULL`, the function will treat the input dataframe as a single epoch.

If the number of samples in one segment is less than 90 samples, the aggregation result will be -1 (marker of invalid value).

See Also

`aggregate_for_orientation` for aggregating to get accelerometer orientation estimation for each epoch.

Other aggregate functions: `aggregate_for_orientation()`

Examples

```
# sample data
df = sample_raw_accel_data
head(df)

# epoch set to 5 seconds, and method set to "trapz"
aggregate_for_mims(df, epoch = '5 sec', method='trapz')

# epoch set to 1 second, method set to "sum"
aggregate_for_mims(df, epoch = '1 sec', method='sum')

# epoch set to 1 second, and st set to be 1 second before the start time of the data
# so the first segment will only include data for 1 second, therefore the resulted
# aggregated value for the first segment will be -1 (invalid) because the
# samples are not enough. And the second segment starts from 11:00:01, instead
# of 11:00:02 as shown in prior example,
aggregate_for_mims(df, epoch = '1 sec', method='sum', st=df[1,1] - 1)
```

`aggregate_for_orientation` *Aggregate over epoch to get estimated accelerometer orientation.*

Description

`aggregate_for_orientation` returns a dataframe with accelerometer orientations estimated by Mizell, 2003 over each epoch (see `compute_orientation`). The epoch start time will be used as timestamp in the first column.

Usage

```
aggregate_for_orientation(
  df,
  epoch,
  estimation_window = 2,
  unit = "deg",
  st = NULL
)
```

Arguments

`df`	dataframe. Input accelerometer data in mhealth format. First column should be timestamps in POSIXt format.
`epoch`	string. Any format that is acceptable by argument `breaks` in method `cut.POSIXt`.For example, "1 sec", "1 min", "5 secs", "10 mins".
`estimation_window`	number. Duration in seconds to be used to estimate orientation within each epoch. Default is 2 (seconds), as suggested by Mizell, 2003.
`unit`	string. The unit of orientation angles. Can be "deg" (degree) or "rad" (radian). Default is "deg".
`st`	character or POSIXct timestamp. An optional start time you can set to force the epochs generated by referencing this start time. If it is NULL, the function will use the first timestamp in the timestamp column as start time to generate epochs. This is useful when you are processing a stream of data and want to use a common start time for segmenting data. Default is NULL.

Details

This function accepts a dataframe (in mhealth accelerometer data format) and computes the estimated acclerometer orientations (in x, y, and z angles) over each fixed epoch. The returned dataframe will have the same format as input dataframe, including four columns, and have the same datetime format as input dataframe in the timestamp column. The orientation estimation method used in the function is based on Mizell, 2003.

Value

dataframe. The returned dataframe will have the same format as input dataframe.

How is it used in mims-unit algorithm?

This function is used in mims-unit algorithm after extrapolation (`extrapolate`). The extrapolated signal will be estimated to get orientation angles using this function.

Note

If epoch argument is not provided or is `NULL`, the function will treat the input dataframe as a single epoch.

If the number of samples in an epoch is less than 90 would be NaN (invalid) for this epoch.

See Also

aggregate_for_mims for aggregating to get integrated values for each axis for each epoch.

Other aggregate functions: aggregate_for_mims()

Examples

```
# Use sample input data
df = sample_raw_accel_data
head(df)

# set epoch to 1 second and unit to degree
# last epoch does not have enough samples to estimate orientation angles.
aggregate_for_orientation(df, epoch='1 sec', unit='deg')

# set epoch to 2 seconds and unit to radian
# last epoch does not have enough samples to estimate orientation angles.
aggregate_for_orientation(df, epoch='2 sec', unit='rad')

# epoch set to 2 seconds, and st set to be 1 second before the start time of the data
# so the first segment will only include data for 1 second, therefore the resulted
# aggregated value for the first segment will be -1 (invalid) because the
# samples are not enough. And the second segment starts from 11:00:01, instead
# of 11:00:01 as shown in prior example,
aggregate_for_orientation(df, epoch = '1 sec', unit='rad', st=df[1,1] - 1)
```

---

bandlimited_interp — *Apply a bandlimited interpolation filter to the signal to change the sampling rate*

Description

bandlimited_interp function takes a multi-channel signal and applies a bandlimited interpolation filter to the signal to change its sampling rate.

Usage

```
bandlimited_interp(df, orig_sr, new_sr)
```

Arguments

df	dataframe. The input multi-channel signal. The first column is timestamps in POSXlct format. The rest columns are signal values.
orig_sr	number. Sampling rate in Hz of the input signal.
new_sr	number. The desired sampling rate in Hz of the output signal.

Details

This function filters the input multi-channel signal by applying a bandlimited interpolation filter. See `resample` for the underlying implementation.

Value

dataframe. Filtered signal.

How is it used in MIMS-unit algorithm?

This function is not used in the released version of MIMS-unit algorithm, but has once been considered to be used after extrapolation to harmonize sampling rate before filtering. But in the end, we decided to use linear interpolation before extrapolation to increase the sampling rate to 100Hz, so this method is no longer needed.

See Also

Other filtering functions: `bessel()`, `iir()`, `remove_average()`

Examples

```
# Use sample data
df = sample_raw_accel_data

# View input
illustrate_signal(df, plot_maxed_out_line = FALSE)

# Apply filtering that uses the same setting as in MIMSunit algorithm
output = bandlimited_interp(df, orig_sr=80, new_sr=30)

# View output
illustrate_signal(output, plot_maxed_out_line = FALSE)
```

---

bessel *Apply Bessel lowpass filter to the signal*

Description

`bessel` function takes a multi-channel signal and applies a bessel lowpass filter to the signal.

Usage

```
bessel(df, sr, cutoff_freq, order = 8)
```

Arguments

`df`	dataframe. The input multi-channel signal. The first column is timestamps in POSXlct format. The rest columns are signal values.
`sr`	number. Sampling rate in Hz of the input signal.
`cutoff_freq`	number. The lowpass cutoff frequency in Hz.
`order`	number. The order of the filter. Default is 8.

Details

This function filters the input multi-channel signal by applying a bessel lowpass filter. See wiki for the explanation of the filter. The filter was implemented with the same implementation as in MATLAB.

Value

dataframe. Filtered signal.

How is it used in MIMS-unit algorithm?

This function has been considered as one of filtering options during the development of MIMS-unit algorithm. But the released version of MIMS-unit algorithm does not use this function for filtering.

See Also

Other filtering functions: `bandlimited_interp()`, `iir()`, `remove_average()`

Examples

```
# Use sample data
df = sample_raw_accel_data

# View input
illustrate_signal(df, plot_maxed_out_line = FALSE)

# Apply filtering
output = bessel(df, sr=80, cutoff_freq=2, order=8)

# View output
illustrate_signal(output, plot_maxed_out_line = FALSE)
```

clip_data — *Clip dataframe to the given start and stop time*

Description

`clip_data` clips the input sensor dataframe according to the given start and stop time

Usage

```
clip_data(df, start_time, stop_time)
```

Arguments

`df`	dataframe. Input dataframe of the multi-channel signal. The first column is the timestamps in POSXlct format and the following columns are accelerometer values.
`start_time`	POSXlct format or character. Start time for clipping. If it is a character, it should be recognizable by as.POSXlct function.
`stop_time`	POSXlct format or character. Stop time for clipping. If it is a character, it should be recognizable by as.POSXlct function.

Details

This function accepts a dataframe of multi-channel signal, clips it according to the start_time and stop_time.

Value

dataframe. The same format as the input dataframe.

How is it used in MIMS-unit algorithm?

This function is a utility function that was used in various part in the algorithm whenever we need to clip a dataframe.

See Also

Other utility functions: `cut_off_signal()`, `interpolate_signal()`, `parse_epoch_string()`, `sampling_rate()`, `segment_data()`, `simulate_new_data()`

Examples

```
default_ops = options()
options(digits.secs=3)
# Use the provided sample data
df = sample_raw_accel_data

# Check the start time and stop time of the dataset
```

```
summary(df)

# Use timestamp string to clip 1 second data
start_time = "2016-01-15 11:01:00"
stop_time = "2016-01-15 11:01:01"
output = clip_data(df, start_time, stop_time)
summary(output)

# Use POSIXct timestamp to clip data
start_time = as.POSIXct("2016-01-15 11:01:00")
stop_time = as.POSIXct("2016-01-15 11:01:01")
output = clip_data(df, start_time, stop_time)
summary(output)

# If start and stop time is not in the range of the input data
# return empty data.frame
start_time = "2016-01-15 12:01:00"
stop_time = "2016-01-15 12:01:01"
output = clip_data(df, start_time, stop_time)
output

# Restore original options
options(default_ops)
```

`compute_orientation` *Estimate the accelerometer orientation*

Description

`compute_orientation` returns a dataframe with accelerometer orientations estimated by Mizell, 2003 for the input dataframe.

Usage

```
compute_orientation(df, estimation_window = 2, unit = "deg")
```

Arguments

`df`	dataframe. Input multi-channel signal. First column should be timestamps in POSIXt format.
`estimation_window`	number. window size in seconds to be used to estimate orientations. Default is 2 (seconds), as suggested by Mizell, 2003.
`unit`	string. The unit of orientation angles. Can be "deg" (degree) or "rad" (radian). Default is "deg".

Details

This function accepts a dataframe (in mhealth accelerometer data format) and computes the estimated acclerometer orientations (in x, y, and z angles) for every `estimation_window` seconds of the entire sequence, and outputs the mean of these angles. The returned dataframe will have the same format as input dataframe, including four columns, and have the same datetime format as input dataframe in the timestamp column. The orientation estimation method used in the function is based on Mizell, 2003.

Value

dataframe. The returned dataframe will have the same format as input dataframe.

How is it used in mims-unit algorithm?

This function is used in function (`aggregate_for_orientation`).

See Also

Other transformation functions: `sum_up()`, `vector_magnitude()`

Examples

```
# Use first 10 second sample data for testing
df = sample_raw_accel_data
df = clip_data(df, start_time = df[1,1], stop_time = df[1, 1] + 600)

# compute orientation angles in degrees
compute_orientation(df)

# compute orientation angles in radian angles
compute_orientation(df, unit='rad')
```

`conceptual_diagram_data`

The input accelerometer data used to generate the conceptual diagram (Figure 1) in the manuscript.

Description

The dataset includes accelerometer data from four devices. Device 0 is a real Actigraph GT9X device configured at 80Hz and 8g. Device 1 to 3 are simulated data from the data of device 0 using function `simulate_new_data`. Data for device 0 is a random selected nondominant wrist data from a participant doing Jumping jack. The data is manipulated to insert an artificial impulse to demonstrate the effect of the MIMS-unit algorithm when dealing on it.

Usage

`conceptual_diagram_data`

Format

A data frame with 1704 rows and 5 variables:

HEADER_TIME_STAMP The timestamp of raw accelerometer data, in POSIXct

X The x axis value of raw accelerometer data, in number

GRANGE The dynamic range of the simulated device in g, in number

SR The sampling rate in Hz of the simulated device, in number

NAME An alternative name that is friendly for plotting for different devices, in character

Source

https://github.com/qutang/MIMSunit

---

custom_mims_unit *Compute both MIMS-unit and sensor orientations with custom settings*

Description

custom_mims_unit computes the Monitor Independent Motion Summary unit and estimates the sensor orientations for the input multi-channel accelerometer signal with custom settings. The input signal can be from devices of any sampling rate and dynamic range. Please refer to the manuscript for detailed description of the algorithm. Please refer to functions for the intermediate steps: extrapolate for extrapolation, iir for filtering, aggregate_for_mims and aggregate_for_orientation for aggregation.

Usage

```
custom_mims_unit(
  df,
  epoch = "5 sec",
  dynamic_range,
  noise_level = 0.03,
  k = 0.05,
  spar = 0.6,
  filter_type = "butter",
  cutoffs = c(0.2, 5),
  axes = c(2, 3, 4),
  use_extrapolation = TRUE,
  use_filtering = TRUE,
  combination = "sum",
  allow_truncation = TRUE,
  output_mims_per_axis = FALSE,
  output_orientation_estimation = FALSE,
  epoch_for_orientation_estimation = NULL,
  before_df = NULL,
  after_df = NULL,
```

```
  use_gui_progress = FALSE,
  st = NULL
)
```

Arguments

`df` dataframe. Input multi-channel accelerometer signal.

`epoch` string. Any format that is acceptable by argument `breaks` in method `cut.POSIXt`.For example, "1 sec", "1 min", "5 sec", "10 min". Default is "5 sec".

`dynamic_range` numerical vector. The dynamic ranges of the input signal. Should be a 2-element numerical vector. `c(low,high)`, where `low` is the negative max value the device can reach and `high` is the positive max value the device can reach.

`noise_level` number. The tolerable noise level in g unit, should be between 0 and 1. Default is 0.03, which applies to most devices.

`k` number. Duration of neighborhood to be used in local spline regression for each side, in seconds. Default is 0.05, as optimized by MIMS-unit algorithm.

`spar` number. Between 0 and 1, to control how smooth we want to fit local spline regression, 0 is linear and 1 matches all local points. Default is 0.6, as optimized by MIMS-unit algorithm.

`filter_type` string. The type of filter to be applied. Could be 'butter' for butterworth bandpass filter, 'ellip' for elliptic bandpass filter or 'bessel' for bessel lowpass filter + average removal highpass filter. Default is "butter".

`cutoffs` numerical vector. Cut off frequencies to be used in filtering. If `filter_type` is "bessel", the cut off frequency for lowpass filter would be multiplied by 2 when being used. Default is 0.2Hz and 5Hz.

`axes` numerical vector. Indices of columns that specifies the axis values of the input signal. Default is `c(2,3,4)`.

`use_extrapolation` logical. If it is TRUE, the function will apply extrapolation algorithm to the input signal, otherwise it will skip extrapolation but only linearly interpolate the signal to 100Hz. Default is TRUE.

`use_filtering` logical. If it is TRUE, the function will apply bandpass filtering to the input signal, otherwise it will skip the filtering. Default is TRUE.

`combination` string. Method to combine MIMS-unit values for each axis. Could be "sum" for `sum_up` or "vm" for `vector_magnitude`.

`allow_truncation` logical. If it is TRUE, the algorithm will truncate very small MIMS-unit values to zero. Default is TRUE.

`output_mims_per_axis` logical. If it is TRUE, the output MIMS-unit dataframe will have MIMS-unit values for each axis from the third column. Default is FALSE.

`output_orientation_estimation` logical. If it is TRUE, the function will also estimate sensor orientations over each epoch. And the output will be a list, with the first element being the MIMS-unit dataframe, and the second element being the sensor orientation dataframe. Default is FALSE.

epoch_for_orientation_estimation
: string. string. Any format that is acceptable by argument breaks in method cut.POSIXt.For example, "1 sec", "1 min", "5 sec", "10 min". Default is "5 sec". It is independent from epoch for MIMS-unit.

before_df
: dataframe. The multi-channel accelerometer signal comes before the input signal to be prepended to the input signal during computation. This is used to eliminate the edge effect during extrapolation and filtering. If it is NULL, algorithm will run directly on the input signal. Default is NULL.

after_df
: dataframe. The multi-channel accelerometer signal comes after the input signal to be append to the input signal. This is used to eliminate the edge effect during extrapolation and filtering. If it is NULL, algorithm will run directly on the input signal. Default is NULL.

use_gui_progress
: logical. If it is TRUE, show GUI progress bar on windows platform. Default is FALSE.

st
: character or POSIXct timestamp. An optional start time you can set to force the epochs generated by referencing this start time. If it is NULL, the function will use the first timestamp in the timestamp column as start time to generate epochs. This is useful when you are processing a stream of data and want to use a common start time for segmenting data. Default is NULL.

Value

dataframe or list. If output_orientation_estimation is TRUE, the output will be a list, otherwise the output will be the MIMS-unit dataframe.

The first element will be the MIMS-unit dataframe, in which the first column is the start time of each epoch in POSIXct format, and the second column is the MIMS-unit value for the input signal, and the third column and on are the MIMS-unit values for each axis of the input signal if output_mims_per_axis is TRUE.

The second element will be the orientation dataframe, in which the first column is the start time of each epoch in POSIXct format, and the second to fourth column is the estimated orientations for the input signal.

How is it used in MIMS-unit algorithm?

This is the low-level entry of MIMS-unit and orientation estimation algorithm. mims_unit calls this function internally.

Note

This function allows you to run customized algorithm for MIMSunit and sensor orientations.

before_df and after_df are often set when the accelerometer data are divided into files of smaller chunk.

See Also

Other Top level API functions: mims_unit(), sensor_orientations(), shiny_app()

Examples

```
# Use sample data for testing
df = sample_raw_accel_data

# compute mims unit values with custom parameter
output = custom_mims_unit(df, epoch = '1 sec', dynamic_range=c(-8, 8), spar=0.7)
head(output)
```

---

cut_off_signal — *Cut off input multi-channel signal according to a new dynamic range*

Description

`cut_off_signal` cuts off the input multi-channel accelerometer data according to a new dynamic range, then adds gausian noise to the cut-off samples.

Usage

```
cut_off_signal(df, range = NULL, noise_std = 0.03)
```

Arguments

`df`	dataframe. Input multi-channel accelerometer data.
`range`	numerical vector. The new dynamic ranges to cut off the signal. Should be a 2-element numerical vector. `c(low,high)`, where `low` is the negative max value the device can reach and `high` is the positive max value the device can reach. Default is `NULL`, meaning the function will do nothing but return the input data.
`noise_std`	number. The standard deviation of the added gaussian noise.

Details

This function simulates the behavior that a low dynamic range device is trying to record high intensity movement, where recorded accelerometer signal will be cut off at the dynamic range, but the true movement should have higher acceleration values than the dynamic range. This function also adds gaussian noise to the cut off samples to better simulate the real world situation.

Value

dataframe. The multi-channel accelerometer data with the new dynamic range as specified in `range`.

How is it used in MIMS-unit algorithm?

This function is a utility function that is used to simulate the behaviors of low dynamic range devices during algorithm validation.

See Also

Other utility functions: clip_data(), interpolate_signal(), parse_epoch_string(), sampling_rate(), segment_data(), simulate_new_data()

Examples

```
# Use sample data for testing
df = sample_raw_accel_data

# Show df
illustrate_signal(df, range=c(-8, 8))

# cut off the signal to c(-2, 2)
new_df = cut_off_signal(df, range=c(-2, 2), noise_std=0.03)

# Show new df
illustrate_signal(new_df, range=c(-2, 2))
```

---

cv_different_algorithms

Coefficient of variation values for different acceleration data summary algorithms

---

Description

A dataset containing the coefficient of variation values at different frequencies for the dataset that includes accelerometer measures of different devices on a standard elliptical shaker.

Usage

```
cv_different_algorithms
```

Format

A data frame with 30 rows and 3 variables:

TYPE Accelerometer summary algorithm name, in character

HZ The frequency of the elliptical shaker, in number

COEFF_OF_VARIATION The coefficient of variation values, in number

Source

https://github.com/qutang/MIMSunit-dataset-shaker edge_case *A short snippet of raw accelerometer signal from a device that has ending data maxed out.*

Description

The dataset includes accelerometer data sampled at 80Hz and 6g. This data is used to test the edge case.

Usage

```
edge_case
```

Format

A data frame with 20001 rows and 4 variables:

HEADER_TIME_STAMP The timestamp of raw accelerometer data, in POSIXct

X The x axis value of raw accelerometer data, in number

Y The x axis value of raw accelerometer data, in number

Z The x axis value of raw accelerometer data, in number

Source

https://github.com/qutang/MIMSunit

export_to_actilife *Export accelerometer data in Actilife RAW CSV format*

Description

`export_to_actilife` exports the input dataframe as a csv file that is compatible with Actilife.

Usage

```
export_to_actilife(
  df,
  filepath,
  actilife_version = "6.13.3",
  firmware_version = "1.6.0"
)
```

Arguments

`df` dataframe. Input accelerometer data. The first column is timestamp in POSXlct format, and the rest columns are accelerometer values in g ($9.81m/s^2$).

`filepath` string. The output filepath.

`actilife_version`
string. The Actilife version number to be added to the header. Default is "6.13.3", that was used by the algorithm during development.

`firmware_version`
string. The firmware version number to be added to the header. This is supposed to be the firmware version of the Actigraph devices. We did not see any usage of the number during the computation of Actigraph counts by Actilife, so it may be set with an arbitrary version code seen in any Actigraph devices. We use default version code "1.6.0".

Details

This function takes an input accelerometer dataframe and exports it in Actilife RAW CSV format with a prepended a madeup header. The exported file csv file has compatible header, column names, timestamp format with Actilife and can be imported directly into Actilife software.

Value

No return value.

How is it used in MIMS-unit algorithm?

This function is an utility function that was used to convert validation data into Actilife RAW CSV format so that we can use Actilife to compute Actigraph counts values for these data.

See Also

Other File I/O functions: `import_actigraph_count_csv()`, `import_actigraph_csv_chunked()`, `import_actigraph_csv()`, `import_actigraph_meta()`, `import_activpal3_csv()`, `import_enmo_csv()`, `import_mhealth_csv_chunked()`, `import_mhealth_csv()`

Examples

```
# Use the first 5 rows from sample data
df = sample_raw_accel_data[1:5,]
head(df)

# Save to current path with default mocked actilife and firmware versions
filepath = tempfile()
export_to_actilife(df, filepath)

# The saved file will have the same format as Actigraph csv files
readLines(filepath)

# Cleanup
file.remove(filepath)
```

extrapolate *Extrapolate input multi-channel accelerometer data*

Description

extrapolate applies the extrapolation algorithm to a multi-channel accelerometer data, trying to reconstruct the true movement from the maxed-out samples.

Usage

```
extrapolate(df, ...)

extrapolate_single_col(
  t,
  value,
  range,
  noise_level = 0.03,
  k = 0.05,
  spar = 0.6
)
```

Arguments

df	dataframe. Input multi-channel accelerometer data. Used in extrapolate. The first column should be the date/time
...	see following parameter list.
t	POSIXct or numeric vector. Input index or timestamp sequence Used in extrapolate_single_col.
value	numeric vector. Value vector used in extrapolate_single_col.
range	numeric vector. The dynamic ranges of the input signal. Should be a 2-element numeric vector. c(low,high), where low is the negative max value the device can reach and high is the positive max value the device can reach.
noise_level	number. The tolerable noise level in g unit, should be between 0 and 1. Default is 0.03, which applies to most devices.
k	number. Duration of neighborhood to be used in local spline regression for each side, in seconds. Default is 0.05, as optimized by MIMS-unit algorithm.
spar	number. Between 0 and 1, to control how smooth we want to fit local spline regression, 0 is linear and 1 matches all local points. Default is 0.6, as optimized by MIMS-unit algorithm.

Details

This function first linearly interpolates the input signal to 100Hz, and then applies the extrapolation algorithm (see the manuscript) to recover the maxed-out samples. Maxed-out samples are samples that are cut off because the intensity of the underlying movement exceeds the dynamic range of the device.

`extrapolate` processes a dataframe of a multi-channel accelerometer signal. `extrapolate_single_col` processes a single-channel signal with its timestamps and values specified in the first and second arguments.

Value

`extraplate` returns a dataframe with extrapolated multi-channel signal. `extrapolate_single_col` returns a dataframe with extrapolated single-channel signal, the timestamp col is in numeric values instead of POSIXct format.

How is it used in MIMS-unit algorithm?

This function is the first step during MIMS-unit algorithm, applied before filtering.

See Also

Other extrapolation related functions: `extrapolate_rate()`

Examples

```
# Use the maxed-out data for the conceptual diagram
df = conceptual_diagram_data[
        conceptual_diagram_data['GRANGE'] == 4,
        c("HEADER_TIME_STAMP", "X")]

# Plot input
illustrate_signal(df, range=c(-4, 4))

# Use the default parameter settings as in MIMunit algorithms
# The dynamic range of the input data is -4g to 4g.
output = extrapolate(df, range=c(-4, 4))

# Plot output
illustrate_signal(output, range=c(-4, 4))
```

---

extrapolate_rate *Get extrapolation rate.*

Description

`extrapolate_rate` computes the extrapolation rate given the test signal (maxed out), the true complete signal (no maxed out) and the extrapolated signal.

Usage

```
extrapolate_rate(test_df, true_df, extrap_df)
```

Arguments

`test_df`	dataframe. See details for the input format.
`true_df`	dataframe. See details for the input format.
`extrap_df`	dataframe. See details for the input format.

Details

All three input dataframes will have the same format, with the first column being timestamps in POSXlct format, and the following columns being acceleration values in g.

Value

number. The extrapolation rate value in double format. If extrapolation rate is 1, it means the extrapolated signal recovers as the true signal. If extrapolation rate is between 0 and 1, it means the extrapolation helps reducing the errors caused by signal maxing out. If extrapolation rate is smaller than 0, it means the extrapolation increases the errors caused by signal maxing out (during over extrapolation).

How is it used in MIMS-unit algorithm?

This function is used to compute extrapolation rate during extrapolation parameter optimization. You may see results in Figure 2 of the manuscript.

See Also

Other extrapolation related functions: `extrapolate()`

Examples

```
# Prepare data for test, ground truth
test_df = conceptual_diagram_data[
              conceptual_diagram_data['GRANGE'] == 4,
              c("HEADER_TIME_STAMP", "X")]
true_df = conceptual_diagram_data[
              conceptual_diagram_data['GRANGE'] == 8,
              c("HEADER_TIME_STAMP", "X")]

# Do extrapolation
extrap_df = extrapolate(test_df, range=c(-4, 4))

# Compute extrapolation rate
extrapolate_rate(test_df, true_df, extrap_df)
```

`generate_interactive_plot` *Plot MIMS unit values or raw signal using dygraphs interactive plotting library.*

Description

`generate_interactive_plot` plots MIMS unit values or raw signal using dygraphs interactive plotting library.

Usage

```
generate_interactive_plot(df, y_label, value_cols = c(2, 3, 4))
```

Arguments

`df`	data.frame.The dataframe storing MIMS unit values or raw accelerometer signal. The first column should be timestamps.
`y_label`	str. The label name to be put on the y axis.
`value_cols`	numerical vector. The indices of columns storing values, typically starting from the second column. The default is ‘c(2,3,4)‘.

Value

A dygraphs graph object. When showing, the graph will be plotted in a html widgets in an opened browser.

See Also

Other visualization functions.: `illustrate_extrapolation()`, `illustrate_signal()`

Examples

```
# Use sample data for testing
df = sample_raw_accel_data

# Plot using default settings, due to pkgdown limitation, no interactive
# plots will be shown on the website page.
generate_interactive_plot(df,
                          y_label="Acceleration (g)")

# The function can be used to plot MIMS unit values as well
mims = mims_unit(df, dynamic_range=c(-8, 8))
generate_interactive_plot(mims,
                          y_label="MIMS-unit values",
                          value_cols=c(2))
```

iir — *Apply IIR filter to the signal*

Description

`iir` function takes a multi-channel signal and applies an IIR filter to the signal.

Usage

```
iir(df, sr, cutoff_freq, order = 4, type = "high", filter_type = "butter")
```

Arguments

`df`	dataframe. The input multi-channel signal. The first column is timestamps in POSXlct format. The rest columns are signal values.
`sr`	number. Sampling rate in Hz of the input signal.
`cutoff_freq`	number or numerical vector. The cutoff frequencies in Hz. If the IIR filter is a bandpass or bandstop filter, it will be a 2-element numerical vector specifying the low and high end cutoff frequencies `c(low,high)`.
`order`	number. The order of the filter. Default is 4.
`type`	string. Filtering type, one of "low" for a low-pass filter, "high" for a high-pass filter, "stop" for a stop-band (band-reject) filter, or "pass" for a pass-band filter.
`filter_type`	string. IIR filter type, one of "butter" for butterworth filter, "chebyI" for Chebyshev Type I filter, or "ellip" for Elliptic filter.

Details

This function filters the input multi-channel signal by applying an IIR filter. See wiki for the explanation of the filter. The implementations of IIR filters can be found in `butter`, `cheby1`, and `ellip`.

For Chebyshev Type I, Type II and Elliptic filter, the passband ripple is fixed to be 0.05 dB. For Elliptic filter, the stopband ripple is fixed to be -50dB.

Value

dataframe. Filtered signal.

How is it used in MIMS-unit algorithm?

This function has been used as the main filtering method in MIMS-unit algorithm. Specifically, it uses a 0.5 - 5 Hz bandpass butterworth filter during filtering.

See Also

Other filtering functions: `bandlimited_interp()`, `bessel()`, `remove_average()`

Examples

```
# Use sample data
df = sample_raw_accel_data

# View input
illustrate_signal(df, plot_maxed_out_line = FALSE)

# Apply filtering that uses the same setting as in MIMSunit algorithm
output = iir(df, sr=80, cutoff_freq=c(0.2, 5), type='pass')

# View output
illustrate_signal(output, plot_maxed_out_line = FALSE)
```

`illustrate_extrapolation` *Plot illustrations about extrapolation in illustration style.*

Description

`illustrate_extrapolation` plots elements of extrapolations (e.g., marked points, reference lines) in the same style as `illustrate_signal`.

Usage

```
illustrate_extrapolation(
  df,
  dynamic_range,
  title = NULL,
  show_neighbors = TRUE,
  show_extrapolated_points_and_lines = TRUE,
  ...
)
```

Arguments

`df`	data.frame.The original data before extrapolation.
`dynamic_range`	numerical vector. The dynamic ranges of the input signal. Should be a 2-element numerical vector. `c(low,high)`, where `low` is the negative max value the device can reach and `high` is the positive max value the device can reach.
`title`	Char. The title of the plot.
`show_neighbors`	bool. Show the points used for extrapolation if TRUE.
`show_extrapolated_points_and_lines`	bool. Show the extrapolated points and curves used for extrapolation.
`...`	Parameters that can be used to tune extrapolation, including `spar`, `k`, and `noise_level`. See `extrapolate` for explanations.

Value

ggplot2 graph object. The graph to be shown.

See Also

Other visualization functions.: generate_interactive_plot(), illustrate_signal()

Examples

```
# Use the maxed-out data for the conceptual diagram
df = conceptual_diagram_data[
       conceptual_diagram_data['GRANGE'] == 2,
       c("HEADER_TIME_STAMP", "X")]

# Plot extrapolation illustration using default settings
illustrate_extrapolation(df, dynamic_range=c(-2,2))

# Do not show neighbor points
illustrate_extrapolation(df, dynamic_range=c(-2,2), show_neighbors=FALSE)

# Do not show extrapolated points and lines
illustrate_extrapolation(df,
                         dynamic_range=c(-2,2),
                         show_extrapolated_points_and_lines=FALSE)
```

`illustrate_signal` *Plot given raw signal in illustration diagram style.*

Description

`illustrate_signal` plots the given uniaxial signal in illustration diagram style. Illustration diagram style hides axes markers, unnecessary guidelines.

Usage

```
illustrate_signal(
  data,
  point_size = 0.3,
  plot_point = TRUE,
  line_size = 0.3,
  plot_line = TRUE,
  range = c(-2, 2),
  plot_maxed_out_line = TRUE,
  plot_origin = TRUE,
  title = NULL,
  plot_title = TRUE
)
```

Arguments

`data`	data.frame. The input uniaxial signal. First column should be timestamp.
`point_size`	number. The size of the plotted data point.
`plot_point`	Bool. Plot signal as points if TRUE.
`line_size`	number. The line width of the plotted signal curve.
`plot_line`	Bool. Plot signal with curve if TRUE.
`range`	vector. Dynamic range of the signal.
`plot_maxed_out_line`	Bool. Plot dynamic range lines if TRUE. Dynamic range is set by 'range'.
`plot_origin`	Bool. Plot the 0 horizontal line if TRUE.
`title`	Char. The title of the plot.
`plot_title`	Bool. Plot title if TRUE.

Value

ggplot2 graph object. The graph to be shown.

See Also

Other visualization functions.: `generate_interactive_plot()`, `illustrate_extrapolation()`

Examples

```
# Use sample data for testing
df = sample_raw_accel_data

# Plot it with default settings
illustrate_signal(df)

# Plot with a different style
illustrate_signal(df, point_size=1, line_size=1)

# Turn off annotation lines
illustrate_signal(df, plot_maxed_out_line = FALSE, plot_origin = FALSE)

# Use title
illustrate_signal(df, plot_title=TRUE, title = "This is a title")
```

`import_actigraph_count_csv` *Import Actigraph count data stored in Actigraph summary csv format*

Description

`import_actigraph_count_csv` imports Actigraph count data stored in Actigraph summary csv format, which was exported by Actilife.

Usage

```
import_actigraph_count_csv(
  filepath,
  count_col = 2,
  count_per_axis_cols = c(2, 3, 4)
)
```

Arguments

`filepath`	string. The filepath of the input data.
`count_col`	number. The index of column of Actigraph count (combined axes). If it is `NULL`, the function will use `count_per_axis_cols` to get the combined Actigraph count values.
`count_per_axis_cols`	numerical vector. The indices of columns of Actigraph count values per axis. If `count_col` is not `NULL`, the argument will be ignored. If it is `NULL`, the output dataframe will only have two columns without Actigraph count values per axis.

Value

dataframe. The imported actigraph count data, with the first column being the timestamps in POSIXct format, and the second column being the combined Actigraph count values, and the rest of columns being the Actigraph cont values per axis if available. Column names: `HEADER_TIME_STAMP`, `ACTIGRAPH_COUNT`, `ACTIGRAPH_COUNT_X`....

How is it used in MIMS-unit algorithm?

This function is a File IO function that is used to import Actigraph count data from Actigraph devices during algorithm validation.

Note

If both `count_col` and `count_per_axis_cols` are `NULL`, the function will raise an error.

See Also

Other File I/O functions: `export_to_actilife()`, `import_actigraph_csv_chunked()`, `import_actigraph_csv()`, `import_actigraph_meta()`, `import_activpal3_csv()`, `import_enmo_csv()`, `import_mhealth_csv_chunked()`, `import_mhealth_csv()`

Examples

```
# Use the actigraph count csv file shipped with the package
filepath = system.file('extdata', 'actigraph_count.csv', package='MIMSunit')

# Check original data format
readLines(filepath)[1:5]

# Load file, default column for actigraph count values are 2, this file does not have
```

```
# axial count values
output = import_actigraph_count_csv(filepath, count_col=2)

# Check output
head(output)
```

---

import_actigraph_csv *Import raw multi-channel accelerometer data stored in Actigraph raw csv format*

Description

`import_actigraph_csv` imports the raw multi-channel accelerometer data stored in Actigraph raw csv format. It supports files from the following devices: GT3X, GT3X+, GT3X+BT, GT9X, and GT9X-IMU.

Usage

```
import_actigraph_csv(
  filepath,
  in_voltage = FALSE,
  has_ts = TRUE,
  header = TRUE
)
```

Arguments

`filepath`	string. The filepath of the input data.
`in_voltage`	set as TRUE only when the input Actigraph csv file is in analog quantized format and need to be converted into g value
`has_ts`	set as TRUE only when timestamp is provided as the first column
`header`	boolean. If TRUE, the input csv file will have column names in the first row.

Details

For old device (GT3X) that stores accelerometer values as digital voltage. The function will convert the values to g unit using the following equation.

$$x_g = \frac{x_{voltage} r}{(2^r) - \frac{v}{2}}$$

Where v is the max voltage corresponding to the max accelerometer value that can be found in the meta section in the csv file; r is the resolution level which is the number of bits used to store the voltage values. r can also be found in the meta section in the csv file.

Value

dataframe. The imported multi-channel accelerometer signal, with the first column being the timestamps in POSXlct format, and the rest columns being accelerometer values in g unit.

How is it used in MIMS-unit algorithm?

This function is a File IO function that is used to import data from Actigraph devices during algorithm validation.

See Also

Other File I/O functions: export_to_actilife(), import_actigraph_count_csv(), import_actigraph_csv_chunked(), import_actigraph_meta(), import_activpal3_csv(), import_enmo_csv(), import_mhealth_csv_chunked(), import_mhealth_csv()

Examples

```
default_ops = options()
options(digits.secs=3)

# Use the sample actigraph csv file provided by the package
filepath = system.file('extdata', 'actigraph.csv', package='MIMSunit')

# Check file format
readLines(filepath)[1:15]

# Load the file without timestamp column
df = import_actigraph_csv(filepath, has_ts=FALSE)

# Check loaded file
head(df)

# Check more
summary(df)

# If set has_ts wrong, you should see a warning
df = import_actigraph_csv(filepath, has_ts=TRUE)

# Check loaded file
head(df)

# Check more
summary(df)

# Restore default options
options(default_ops)
```

import_actigraph_csv_chunked

Import large raw multi-channel accelerometer data stored in Actigraph raw csv format in chunks

Description

`import_actigraph_csv_chunked` imports the raw multi-channel accelerometer data stored in Actigraph raw csv format. It supports files from the following devices: GT3X, GT3X+, GT3X+BT, GT9X, and GT9X-IMU.

Usage

```
import_actigraph_csv_chunked(
  filepath,
  in_voltage = FALSE,
  has_ts = TRUE,
  header = TRUE,
  chunk_samples = 180000
)
```

Arguments

`filepath`	string. The filepath of the input data.
`in_voltage`	set as TRUE only when the input Actigraph csv file is in analog quantized format and need to be converted into g value
`has_ts`	set as TRUE only when timestamp is provided as the first column
`header`	boolean. If TRUE, the input csv file will have column names in the first row.
`chunk_samples`	number. The number of samples in each chunk. Default is 180000.

Details

For old device (GT3X) that stores accelerometer values as digital voltage. The function will convert the values to g unit using the following equation.

$$x_g = \frac{x_{voltage} r}{(2^r) - \frac{v}{2}}$$

Where v is the max voltage corresponding to the max accelerometer value that can be found in the meta section in the csv file; r is the resolution level which is the number of bits used to store the voltage values. r can also be found in the meta section in the csv file.

Value

list. The list contains two items. The first item is a generator function that each time it is called, it will return a data.frame of the imported chunk. The second item is a `close` function which you can call at any moment to close the file loading.

How is it used in MIMS-unit algorithm?

This function is a File IO function that is used to import data from Actigraph devices during algorithm validation.

See Also

Other File I/O functions: `export_to_actilife()`, `import_actigraph_count_csv()`, `import_actigraph_csv()`, `import_actigraph_meta()`, `import_activpal3_csv()`, `import_enmo_csv()`, `import_mhealth_csv_chunked()`, `import_mhealth_csv()`

Examples

```
default_ops = options()
options(digits.secs=3)

# Use the actigraph csv file shipped with the package
filepath = system.file('extdata', 'actigraph.csv', package='MIMSunit')

# Check original file format
readLines(filepath)[1:15]

# Example 1: Load chunks every 2000 samples
results = import_actigraph_csv_chunked(filepath, has_ts=FALSE, chunk_samples=2000)
next_chunk = results[[1]]
close_connection = results[[2]]
# Check data as chunks, you can see chunks are shifted at each iteration.
n = 1
repeat {
  df = next_chunk()
  if (nrow(df) > 0) {
    print(paste('chunk', n))
    print(paste("df:", df[1, 1], '-', df[nrow(df),1]))
    n = n + 1
  }
  else {
    break
  }
}

# Close connection after reading all the data
close_connection()

# Example 2: Close loading early
results = import_actigraph_csv_chunked(filepath, has_ts=FALSE, chunk_samples=2000)
next_chunk = results[[1]]
close_connection = results[[2]]
# Check data as chunks, you can see chunk time is shifting forward at each iteration.
n = 1
repeat {
  df = next_chunk()
  if (nrow(df) > 0) {
    print(paste('chunk', n))
```

```
    print(paste("df:", df[1, 1], '-', df[nrow(df),1]))
    n = n + 1
    close_connection()
  }
  else {
    break
  }
}

# Restore default options
options(default_ops)
```

---

import_actigraph_meta *Import The meta information stored in Actigraph RAW or summary csv file.*

Description

`import_actigraph_meta` imports meta information stored in the Actigraph summary csv file.

Usage

```
import_actigraph_meta(filepath, header = TRUE)
```

Arguments

`filepath`	string. The filepath of the input data.
`header`	logical. Whether the Actigraph RAW or summary csv file includes column names. Default is TRUE.

Details

The returned meta information includes following fields.

- sr: Sampling rate in Hz.
- fw: Firmware version. For example "1.7.0".
- sw: Software version of Actilife. For example "6.13.0".
- sn: Serial number of the device.
- st: Start time of the data, in POSIXct format.
- dt: Download time of the data, in POSIXct format.
- at: Type of the device. Could be "MAT","CLE", "MOS" or "TAS", corresponding to different Actigraph devices.
- imu: Whether the file is about Actigraph GT9X IMU data.
- gr: The dynamic range in g unit.
- vs: The voltage level of the device, may be used in AD conversion. See `import_actigraph_csv`.
- res: The resolution or the number of bits used to store quantized voltage values of the device, may be used in AD conversion. See `import_actigraph_csv`.

Value

list. A list of Actigraph device meta information.

How is it used in MIMS-unit algorithm?

This function is a File IO function that is used to get related meta information such as sampling rate, firmware version from Actigraph devices.

See Also

Other File I/O functions: `export_to_actilife()`, `import_actigraph_count_csv()`, `import_actigraph_csv_chunked()`, `import_actigraph_csv()`, `import_activpal3_csv()`, `import_enmo_csv()`, `import_mhealth_csv_chunked()`, `import_mhealth_csv()`

Examples

```
default_ops = options()
options(digits.secs=3)

# Use the sample actigraph csv file provided by the package
filepath = system.file('extdata', 'actigraph.csv', package='MIMSunit')

# Check file format
readLines(filepath)[1:15]

# Load the meta headers of input file
import_actigraph_meta(filepath, header=TRUE)

# Restore default options
options(default_ops)
```

---

import_activpal3_csv — *Import raw multi-channel accelerometer data stored in ActivPal3 csv format*

Description

`import_activpal3_csv` imports the raw multi-channel accelerometer data stored in ActivPal3 csv format by converting the accelerometer values (in digital voltage values) to g unit.

Usage

```
import_activpal3_csv(filepath, header = FALSE)
```

Arguments

`filepath`	string. The filepath of the input data.
`header`	boolean. If TRUE, the input csv file will have column names in the first row.

Details

ActivPal 3 sensors have known dynamic range to be $(-2g, +2g)$. And the sensor stores values using 8-bit memory storage. So, the digital voltage values may be converted to g unit using following equation.

$$x_g = \frac{x_{voltage} - 127}{2^8} * 4$$

Value

dataframe. The imported multi-channel accelerometer signal, with the first column being the timestamps in POSXlct format, and the rest columns being accelerometer values in g unit.

How is it used in MIMS-unit algorithm?

This function is a File IO function that is used to import data from ActivPal3 devices during algorithm validation.

See Also

Other File I/O functions: export_to_actilife(), import_actigraph_count_csv(), import_actigraph_csv_chunked(), import_actigraph_csv(), import_actigraph_meta(), import_enmo_csv(), import_mhealth_csv_chunked(), import_mhealth_csv()

Examples

```
default_ops = options()
options(digits.secs=3)
# Use the sample activpal3 csv file provided by the package
filepath = system.file('extdata', 'activpal3.csv', package='MIMSunit')

# Check the csv format
readLines(filepath)[1:5]

# Load the file, in our case without header
df = import_activpal3_csv(filepath, header=FALSE)

# Check loaded file
head(df)

# Check more
summary(df)

# Restore default options
options(default_ops)
```

import_enmo_csv *Import ENMO data stored in csv csv*

Description

import_enmo_csv imports ENMO data stored in a summary csv format, which was exported by the biobank data analysis tools.

Usage

```
import_enmo_csv(filepath, enmo_col = 2)
```

Arguments

filepath	string. The filepath of the input data.
enmo_col	number. The index of column of ENMO values in the csv file.

Value

dataframe. The imported ENMO data, with the first column being the timestamps in POSIXct format, and the second column being the ENMO values. Column names: HEADER_TIME_STAMP, ENMO.

How is it used in MIMS-unit algorithm?

This function is a File IO function that is used to import ENMO data from activity monitor devices during algorithm validation.

See Also

Other File I/O functions: export_to_actilife(), import_actigraph_count_csv(), import_actigraph_csv_chunked(), import_actigraph_csv(), import_actigraph_meta(), import_activpal3_csv(), import_mhealth_csv_chunked(), import_mhealth_csv()

Examples

```
# Use the enmo csv file shipped with the package
filepath = system.file('extdata', 'enmo.csv', package='MIMSunit')

# Check original data format
readLines(filepath)[1:5]

# Load file, default column for enmo values are 2
output = import_enmo_csv(filepath, enmo_col=2)

# Check output
head(output)
```

import_mhealth_csv	*Import raw multi-channel accelerometer data stored in mHealth Specification*

Description

import_mhealth_csv imports the raw multi-channel accelerometer data stored in mHealth Specification. Note that this function will fail when loading data that have size too big to fit in the memory. For large data file, please use import_mhealth_csv_chunked to load.

Usage

```
import_mhealth_csv(filepath)
```

Arguments

filepath	string. The filepath of the input data.

Value

dataframe. The imported multi-channel accelerometer signal, with the first column being the timestamps in POSXlct format, and the rest columns being accelerometer values in g unit.

How is it used in MIMS-unit algorithm?

This function is a File IO function that is used to import data stored in mHealth Specification during algorithm validation.

See Also

Other File I/O functions: export_to_actilife(), import_actigraph_count_csv(), import_actigraph_csv_chunked(), import_actigraph_csv(), import_actigraph_meta(), import_activpal3_csv(), import_enmo_csv(), import_mhealth_csv_chunked()

Examples

```
default_ops = options()
options(digits.secs=3)
# Use the sample mhealth csv file provided by the package
filepath = system.file('extdata', 'mhealth.csv', package='MIMSunit')
filepath

# Load the file
df = import_mhealth_csv(filepath)

# Check loaded file
head(df)

# Check more
```

```
summary(df)

# Restore default options
options(default_ops)
```

---

import_mhealth_csv_chunked

Import large raw multi-channel accelerometer data stored in mHealth Specification in chunks.

Description

import_mhealth_csv_chunked imports the raw multi-channel accelerometer data stored in mHealth Specification in chunks.

Usage

```
import_mhealth_csv_chunked(filepath, chunk_samples = 180000)
```

Arguments

filepath	string. The filepath of the input data.
chunk_samples	number. The number of samples in each chunk. Default is 180000, which is half hour data for 100 Hz sampling rate.

Value

list. The list contains two items. The first item is a generator function that each time it is called, it will return a dataframe with at most chunk_samples samples of imported data. The third item is a close_connection function which you can call at any moment to close the file loading.

How is it used in MIMS-unit algorithm?

This function is a File IO function that is used to import data stored in mHealth Specification during algorithm validation.

See Also

Other File I/O functions: export_to_actilife(), import_actigraph_count_csv(), import_actigraph_csv_chunked(), import_actigraph_csv(), import_actigraph_meta(), import_activpal3_csv(), import_enmo_csv(), import_mhealth_csv()

Examples

```
default_ops = options()
options(digits.secs=3)

# Use the mhealth csv file shipped with the package
filepath = system.file('extdata', 'mhealth.csv', package='MIMSunit')

# Example 1
# Load chunks every 1000 samples
results = import_mhealth_csv_chunked(filepath, chunk_samples=100)
next_chunk = results[[1]]
close_connection = results[[2]]
# Check data as chunks, you can see chunk time is shifting forward at each iteration.
n = 1
repeat {
  df = next_chunk()
  if (nrow(df) > 0) {
    print(paste('chunk', n))
    print(paste("df:", df[1, 1], '-', df[nrow(df),1]))
    n = n + 1
  } else {
    break
  }
}

# Close connection after reading all the data
close_connection()

# Example 2: close loading early
results = import_mhealth_csv_chunked(filepath, chunk_samples=1000)
next_chunk = results[[1]]
close_connection = results[[2]]
# Check data as chunks, you can see chunk time is shifting forward at each iteration.
n = 1
repeat {
  df = next_chunk()
  if (nrow(df) > 0) {
    print(paste('chunk', n))
    print(paste("df:", df[1, 1], '-', df[nrow(df),1]))
    n = n + 1
    close_connection()
  }
  else {
    break
  }
}

# Restore default options
options(default_ops)
```

`interpolate_signal` *Interpolate missing points and unify sampling rate for multi-channel signal*

Description

`interpolate_signal` applies different interpolation algorithms to the input multi-channel signal to fill in the missing samples and harmonizes the sampling rate.

Usage

```
interpolate_signal(
  df,
  method = "spline_natural",
  sr = 100,
  st = NULL,
  et = NULL
)
```

Arguments

`df`	dataframe. Input multi-channel accelerometer signal.
`method`	string. Interpolation algorithms. Could be "spline_natural", "spline_improved" or "spline_fmm": see `splinefun`; and "linear": see `approxfun`. Default is "spline_natural".
`sr`	number. Sampling rate in Hz of the output signal. Default is 100.
`st`	POSIXct date. The start time for interpolation. If it is `NULL`, it will use the start time of the input signal. Default is `NULL`.
`et`	POSIXct date. The end time for interpolation. If it is `NULL`, it will use the end time of the input signal. Default is `NULL`.

Value

dataframe. Interpolated signal.

How is it used in MIMS-unit algorithm?

This function is a utility function that has been used in functions: `extrapolate`, and `simulate_new_data`.

See Also

Other utility functions: `clip_data()`, `cut_off_signal()`, `parse_epoch_string()`, `sampling_rate()`, `segment_data()`, `simulate_new_data()`

Examples

```
# Use sample data
df = sample_raw_accel_data

# Plot input
illustrate_signal(df, plot_maxed_out_line=FALSE)

# Interpolate to 100 Hz
sr = 100

# Interpolate the entire sequence of data
output = interpolate_signal(df, sr=sr)

# Plot output
illustrate_signal(output, plot_maxed_out_line=FALSE)

# Interpolate part of the sequence
output = interpolate_signal(df, sr=sr, st=df[10,1], et=df[100,1])

# Plot output
illustrate_signal(output, plot_maxed_out_line=FALSE)
```

---

measurements_different_devices

The mean and standard deviation of accelerometer summary measure for different acceleration data summary algorithms and for different devices.

Description

A dataframe contains the mean and standard deviation of accelerometer summary measured at different frequencies for the raw accelerometer signals from different devices collected from on a standard elliptical shaker.

Usage

```
measurements_different_devices
```

Format

A data frame with 235 rows and 8 variables:

DEVICE The name of different devices, in character

GRANGE The dynamic range of the device in g, in number

SR The sampling rate in Hz of the device, in number

TYPE Accelerometer summary algorithm name, in character

HZ The frequency of the elliptical shaker, in number

NAME An alternative name that is friendly for plotting for devices, in character

mean The mean values of accelerometer summary measure, in number

sd The standard deviation values of accelerometer summary measure, in number

Source

https://github.com/qutang/MIMSunit-dataset-shaker

`mims_unit` *Compute Monitor Independent Motion Summary unit (MIMS-unit)*

Description

`mims_unit` computes the Monitor Independent Motion Summary unit for the input multi-channel accelerometer signal. The input signal can be from devices of any sampling rate and dynamic range. Please refer to the manuscript for detailed description of the algorithm. Please refer to functions for the intermediate steps: `extrapolate` for extrapolation, `iir` for filtering, `aggregate_for_mims` for aggregation.

Usage

```
mims_unit(
  df,
  before_df = NULL,
  after_df = NULL,
  epoch = "5 sec",
  dynamic_range,
  output_mims_per_axis = FALSE,
  use_gui_progress = FALSE,
  st = NULL
)

mims_unit_from_files(
  files,
  epoch = "5 sec",
  dynamic_range,
  output_mims_per_axis = FALSE,
  use_gui_progress = FALSE,
  file_type = "mhealth",
  ...
)
```

Arguments

`df`	dataframe. Input multi-channel accelerometer signal. The first column should be the time component

`before_df` dataframe. The multi-channel accelerometer signal comes before the input signal to be prepended to the input signal during computation. This is used to eliminate the edge effect during extrapolation and filtering. If it is `NULL`, algorithm will run directly on the input signal. Default is NULL.

`after_df` dataframe. The multi-channel accelerometer signal comes after the input signal to be append to the input signal. This is used to eliminate the edge effect during extrapolation and filtering. If it is `NULL`, algorithm will run directly on the input signal. Default is NULL.

`epoch` string. Any format that is acceptable by argument `breaks` in method `cut.POSIXt`.For example, "1 sec", "1 min", "5 sec", "10 min". Default is "5 sec".

`dynamic_range` numerical vector. The dynamic ranges of the input signal. Should be a 2-element numerical vector. `c(low,high)`, where `low` is the negative max value the device can reach and `high` is the positive max value the device can reach.

`output_mims_per_axis`
logical. If it is TRUE, the output MIMS-unit dataframe will have MIMS-unit values for each axis from the third column. Default is FALSE.

`use_gui_progress`
logical. If it is TRUE, show GUI progress bar on windows platform. Default is FALSE.

`st` character or POSIXct timestamp. An optional start time you can set to force the epochs generated by referencing this start time. If it is NULL, the function will use the first timestamp in the timestamp column as start time to generate epochs. This is useful when you are processing a stream of data and want to use a common start time for segmenting data. Default is NULL.

`files` character vector. A list of csv filepaths for raw accelerometer data organized in order to be processed. The data should be consecutive in timestamps. A typical case is a set of hourly or daily files for continuous accelerometer sampling. For a single file, please wrap the filepath in a vector ‘c(filepath)‘.

`file_type` character. "mhealth" or "actigraph". The type of the csv files that store the raw accelerometer data.

`...` additional parameters passed to the import function when reading in the data from the files.

Value

dataframe. The MIMS-unit dataframe. The first column is the start time of each epoch in POSIXct format. The second column is the MIMS-unit value for the input signal. If `output_mims_per_axis` is TRUE, the third column and then are the MIMS-unit values for each axis of the input signal.

How is it used in MIMS-unit algorithm?

This is the main entry of MIMS-unit algorithm.

Note

This function is a wrapper function for the low-level `custom_mims_unit` function. It has set internal parameters as described in the manuscript. If you want to run customized algorithm for MIMSunit or if you want to develop better algorithms based on MIMS-unit algorithm, please use function `custom_mims_unit` where all parameters are tunable.

`before_df` and `after_df` are often set when the accelerometer data are divided into files of smaller chunk.

See Also

Other Top level API functions: `custom_mims_unit()`, `sensor_orientations()`, `shiny_app()`

Examples

```
# Use sample data for testing
df = sample_raw_accel_data

# compute mims unit values and output axial values
output = mims_unit(df, epoch = '1 sec', dynamic_range=c(-8, 8), output_mims_per_axis=TRUE)
head(output)
# Use sample mhealth file for testing
filepaths = c(
  system.file('extdata', 'mhealth.csv', package='MIMSunit')
)

# Test with multiple files
output = mims_unit_from_files(filepaths, epoch = "1 sec", dynamic_range = c(-8, 8))
head(output)
```

---

parse_epoch_string — *Parse epoch string to the corresponding number of samples it represents.*

Description

`parse_epoch_string` parses the epoch string (e.g. "1 min"), and outputs the corresponding number of samples it represents.

Usage

```
parse_epoch_string(epoch_str, sr)
```

Arguments

`epoch_str`	string. The input epoch str as accepted by `breaks` argument of `cut.POSIXt`.
`sr`	number. The sampling rate in Hz used to parse the epoch string.

Details

This function parses the given epoch string (e.g. "5 secs") and outputs the corresponding number of samples represented by the epoch string.

Value

number. The number of samples represented by the epoch string.

How is it used in MIMS-unit algorithm?

This function is used in `aggregate_for_mims` function and `mims_unit` function.

See Also

Other utility functions: `clip_data()`, `cut_off_signal()`, `interpolate_signal()`, `sampling_rate()`, `segment_data()`, `simulate_new_data()`

Examples

```
# 1 min with 80 Hz = 4800 samples
parse_epoch_string('1 min', sr=80)

# 30 sec with 30 Hz = 900 samples
parse_epoch_string('30 sec', sr=30)

# 1 hour with 1 Hz = 3600 samples
parse_epoch_string('1 hour', sr=1)

# 1 day with 10 Hz = 864000 samples
parse_epoch_string('1 day', sr=10)
```

---

remove_average *Remove constant component of the signal*

Description

`remove_average` function takes a multi-channel signal and removes the average value over a filtering window.

Usage

```
remove_average(df, sr, order = 0.5)
```

Arguments

`df`	dataframe. The input multi-channel signal. The first column is timestamps in POSXlct format. The rest columns are signal values.
`sr`	number. Sampling rate in Hz of the input signal.
`order`	number. Window size (in seconds) of the filter. Default is 500 ms.

Details

This function filters the input multi-channel signal by removing the average value within each sliding window. The sliding window size is decided by $w = sr * order$.

Value

dataframe. Filtered signal.

How is it used in MIMS-unit algorithm?

This function has been considered as one of filtering options during the development of MIMS-unit algorithm. But the released version of MIMS-unit algorithm does not use this function for filtering.

See Also

Other filtering functions: `bandlimited_interp()`, `bessel()`, `iir()`

Examples

```
# Use sample data
df = sample_raw_accel_data

# View input
illustrate_signal(df, plot_maxed_out_line = FALSE)

# Apply filtering
output = remove_average(df, sr=80, order=0.5)

# View output
illustrate_signal(output, plot_maxed_out_line = FALSE)
```

`rest_on_table` *A short snippet of raw accelerometer signal from a device resting on a table.*

Description

The dataset includes accelerometer data sampled at 80Hz and 6g. This data is used to derive the thresholding.

Usage

```
rest_on_table
```

Format

A data frame with 5000 rows and 4 variables:

HEADER_TIME_STAMP The timestamp of raw accelerometer data, in POSIXct

X The x axis value of raw accelerometer data, in number

Y The x axis value of raw accelerometer data, in number

Z The x axis value of raw accelerometer data, in number

Source

https://github.com/qutang/MIMSunit

`sample_raw_accel_data` *Sample raw accelerometer data*

Description

A raw accelerometer data file contains treadmill data collected from a human subject.

Usage

```
sample_raw_accel_data
```

Format

A data frame with 480 rows and 4 variables:

HEADER_TIME_STAMP Timestamp, in POSIXct

X X axis values, in number

Y Y axis values, in number

Z Z axis values, in number

Source

https://github.com/qutang/MIMSunit

`sampling_rate` *Estimate sampling rate for multi-channel signal*

Description

`sampling_rate` estimates the sampling rate based on the average time intervals between adjacent samples for the input multi-channel signal.

Usage

```
sampling_rate(df)
```

Arguments

`df`	dataframe. Input dataframe of the multi-channel signal. The first column is the timestamps in POSXlct format and the following columns are accelerometer values.

Details

This function accepts a dataframe of multi-channel signal, computes the duration of the sequence, and gets the sampling rate by dividing the number of samples by it.

Value

number. The estimated sampling rate in Hz.

How is it used in MIMS-unit algorithm?

This function is a utility function that was used in various part in the algorithm whenever we need to know the sampling rate.

See Also

Other utility functions: `clip_data()`, `cut_off_signal()`, `interpolate_signal()`, `parse_epoch_string()`, `segment_data()`, `simulate_new_data()`

Examples

```
# Get the test data
df = sample_raw_accel_data

# Default sampling rate is 80Hz
sampling_rate(df)

# Downsample to 30Hz
output = bandlimited_interp(df, 80, 30)
sampling_rate(output)

# Upsampling to 100Hz
output = bandlimited_interp(df, 80, 100)
sampling_rate(output)
```

---

segment_data — *Segment input dataframe into windows as specified by breaks.* `segment_data` *segments the input sensor dataframe into epoch windows with length specified in breaks.*

Description

This function accepts a dataframe of multi-channel signal, segments it into epoch windows with length specified in breaks.

Usage

```
segment_data(df, breaks, st = NULL)
```

Arguments

- `df` dataframe. Input dataframe of the multi-channel signal. The first column is the timestamps in POSXlct format and the following columns are accelerometer values.
- `breaks` character. An epoch length character that can be accepted by cut.breaks function.
- `st` character or POSIXct timestamp. An optional start time you can set to force the breaks generated by referencing this start time. If it is NULL, the function will use the first timestamp in the timestamp column as start time to generate breaks. This is useful when you are processing a stream of data and want to use a common start time for segmenting data. Default is NULL.

Value

dataframe. The same format as the input dataframe, but with an extra column "SEGMENT" in the end specifies the epoch window a sample belongs to.

How is it used in MIMS-unit algorithm?

This function is a utility function that was used in various part in the algorithm whenever we need to segment a dataframe, e.g., before aggregating values over epoch windows.

See Also

Other utility functions: `clip_data()`, `cut_off_signal()`, `interpolate_signal()`, `parse_epoch_string()`, `sampling_rate()`, `simulate_new_data()`

Examples

```
# Use sample data
df = sample_raw_accel_data

# segment data into 1 minute segments
output = segment_data(df, "1 min")

# check the 3rd segment, each segment would have 1 minute data
summary(output[output['SEGMENT'] == 3,])

# segment data into 15 second segments
output = segment_data(df, "15 sec")

# check the 1st segment, each segment would have 15 second data
summary(output[output['SEGMENT'] == 1,])

# segment data into 1 hour segments
output = segment_data(df, "1 hour")

# because the input data has only 15 minute data
# there will be only 1 segment in the output
unique(output['SEGMENT'])
```

```
summary(output)

# use manually set start time
output = segment_data(df, "15 sec", st='2016-01-15 10:59:50.000')

# check the 1st segment, because the start time is 10 seconds before the
# start time of the actual data, the first segment will only include 5 second
# data.
summary(output[output['SEGMENT'] == 1,])
```

`sensor_orientations` *Estimates sensor orientation*

Description

`sensor_orientations` estimates the orientation angles for the input multi-channel accelerometer signal. The input signal can be from devices of any sampling rate and dynamic range. Please refer to function `compute_orientation` for the implementation of the estimation algorithm.

Usage

```
sensor_orientations(
  df,
  before_df = NULL,
  after_df = NULL,
  epoch = "5 sec",
  dynamic_range,
  st = NULL
)
```

Arguments

`df`	dataframe. Input multi-channel accelerometer signal.
`before_df`	dataframe. The multi-channel accelerometer signal comes before the input signal to be prepended to the input signal during computation. This is used to eliminate the edge effect during extrapolation and filtering. If it is `NULL`, algorithm will run directly on the input signal. Default is NULL.
`after_df`	dataframe. The multi-channel accelerometer signal comes after the input signal to be append to the input signal. This is used to eliminate the edge effect during extrapolation and filtering. If it is `NULL`, algorithm will run directly on the input signal. Default is NULL.
`epoch`	string. Any format that is acceptable by argument `breaks` in method `cut.POSIXt`.For example, "1 sec", "1 min", "5 sec", "10 min". Default is "5 sec".
`dynamic_range`	numerical vector. The dynamic ranges of the input signal. Should be a 2-element numerical vector. `c(low,high)`, where `low` is the negative max value the device can reach and `high` is the positive max value the device can reach.

`st` character or POSIXct timestamp. An optional start time you can set to force the epochs generated by referencing this start time. If it is NULL, the function will use the first timestamp in the timestamp column as start time to generate epochs. This is useful when you are processing a stream of data and want to use a common start time for segmenting data. Default is NULL.

Value

dataframe. The orientation dataframe. The first column is the start time of each epoch in POSIXct format. The second to fourth columns are the orientation angles.

How is it used in MIMS-unit algorithm?

This is not included in the official MIMS-unit algorithm nor the manuscript, but we found it is useful to know the sensor orientations in addition to the summary of movement.

Note

This function interpolates and extrapolates the signal before estimating the orientation angles.

`before_df` and `after_df` are often set when the accelerometer data are divided into files of smaller chunk.

See Also

Other Top level API functions: `custom_mims_unit()`, `mims_unit()`, `shiny_app()`

Examples

```
# Use sample data for testing
df = sample_raw_accel_data

# compute sensor orientation angles
sensor_orientations(df, epoch = '2 sec', dynamic_range=c(-8, 8))

# compute sensor orientation angles with different epoch length
output = sensor_orientations(df, epoch = '1 sec', dynamic_range=c(-8, 8))
head(output)
```

---

shiny_app *Run shiny app to compute MIMSunit values from files*

Description

`shiny_app` runs a local shiny app that provides a user friendly interface to compute mims unit values from files stored in mhealth or actigraph format.

Usage

```
shiny_app(options = list())
```

Arguments

`options`	The options passed to `shinyApp`.

How is it used in MIMS-unit algorithm?

This provides a user friendly graphical interface to load local files, call `mims_unit_from_files` and display the results as an interactive graph.

See Also

Other Top level API functions: `custom_mims_unit()`, `mims_unit()`, `sensor_orientations()`

Examples

```
## Not run:
shiny_app()

## End(Not run)
```

`simulate_new_data` *Simulate new data based on the given multi-channel accelerometer data*

Description

`simulate_new_data` simulate new data based on the given multi-channel accelerometer data, a new dynamic range and a new sampling rate.

Usage

```
simulate_new_data(old_data, new_range, new_sr)
```

Arguments

`old_data`	dataframe. Input multi-channel accelerometer data.
`new_range`	numerical vector. The new dynamic ranges to cut off the signal. Should be a 2-element numerical vector. `c(low,high)`, where `low` is the negative max value the device can reach and `high` is the positive max value the device can reach. Default is `NULL`, meaning the function will do nothing but return the input data.
`new_sr`	number. New sampling rate in Hz.

Details

This function simulates the data from a new device based on the signal from a baseline device. It first changes the sampling rate using function `interpolate_signal`, and then changes the dynamic range using function `cut_off_signal`.

How is it used in MIMS-unit algorithm?

This function is a utility function that is used to simulate new devices with different sampling rates and dynamic ranges during algorithm validation.

See Also

Other utility functions: `clip_data()`, `cut_off_signal()`, `interpolate_signal()`, `parse_epoch_string()`, `sampling_rate()`, `segment_data()`

Examples

```
# Use sample data for testing
df = sample_raw_accel_data

# Show df
illustrate_signal(df, range=c(-8, 8))

# simulate new data by changing range and sampling rate
new_df = simulate_new_data(df, new_range=c(-2, 2), new_sr = 30)

# Show new df
illustrate_signal(new_df, range=c(-2, 2))
```

---

sum_up — *Sum of multi-channel signal.*

Description

`sum_up` computes the sum up value for each sample (row) of a multi-channel signal.

Usage

```
sum_up(df, axes = NULL)
```

Arguments

`df`	dataframe. multi-channel signal, with the first column being the timestamp in POSXct format.
`axes`	numerical vector. Specify the column indices for each axis. When this value is NULL, the function assumes the axes are starting from column 2 to the end. Default is `NULL`.

Details

This function takes a dataframe of a multi-channel signal as input, and then computes the sum of each row and returns a transformed dataframe with two columns.

Value

dataframe. The transformed dataframe will have the same number of rows as input dataframe but only two columns, with the first being timestamps and second being the sum up values.

How is it used in MIMS-unit algorithm?

This function is used to combine MIMS-unit values on each axis into a single value after aggregating over each epoch using `aggregate_for_mims`.

See Also

`vector_magnitude`

Other transformation functions: `compute_orientation()`, `vector_magnitude()`

Examples

```
# Use the first 10 rows of the sample data as an example
df = sample_raw_accel_data[1:10,]
df

# By default, the function will assume columns starting from 2 to be axial
# values.
sum_up(df)

# Or, you may specify the column indices yourself
sum_up(df, axes=c(2,3,4))

# Or, if you only want to consider x and y axes
sum_up(df, axes=c(2,3))

# Or, just return the chosen column
sum_up(df, axes=c(2))
```

`vector_magnitude` *Vector magnitude of multi-channel signal.*

Description

`vector_magnitude` computes the vector magnitude value for each sample (row) of a multi-channel signal.

Usage

```
vector_magnitude(df, axes = NULL)
```

Arguments

`df`	dataframe. multi-channel signal, with the first column being the timestamp in POSXct format.
`axes`	numerical vector. Specify the column indices for each axis. When this value is NULL, the function assumes the axes are starting from column 2 to the end. Default is `NULL`.

Details

This function takes a dataframe of a multi-channel signal as input, and then computes the 2-norm (vector magnitude) for each row and returns a transformed dataframe with two columns.

Value

dataframe. The transformed dataframe will have the same number of rows as input dataframe but only two columns, with the first being timestamps and second being the vector magnitude values.

How is it used in MIMS-unit algorithm?

This function is not used in the released version of MIMS-unit algorithm, but was used to compare the alternative `sum_up` method when combining MIMS-unit values on each axis into a single value.

See Also

`sum_up`

Other transformation functions: `compute_orientation()`, `sum_up()`

Examples

```
# Use the first 10 rows of the sample data as an example
df = sample_raw_accel_data[1:10,]
df

# By default, the function will assume columns starting from 2 to be axial
# values.
vector_magnitude(df)

# Or, you may specify the column indices yourself
vector_magnitude(df, axes=c(2,3,4))

# Or, if you only want to consider x and y axes
vector_magnitude(df, axes=c(2,3))

# Or, just return the chosen column
vector_magnitude(df, axes=c(2))
```

Index