

Package: PhysioWearable (via r-universe)

October 3, 2026

Title Free-Living Wearable Accelerometry for the Physio Ecosystem

Version 0.5.5

Description Free-living physical-activity analysis of raw tri-axial accelerometer data, following the GGIR methodology. Computes the Euclidean-Norm-Minus-One (ENMO) acceleration metric with epoch aggregation, gravity-based auto-calibration (van Hees et al. 2014), non-wear detection, step detection, physical-activity intensity classification and bout detection, and a free-living physical-behaviour summary (intensity gradient and MX metrics, activity fragmentation, diurnal profile and guideline attainment) linked to the ICF Activities and Participation performance qualifier. Ingests raw accelerometer exports (GENEActiv / ActiGraph / Axivity CSV) into PhysioExperiment objects.

Depends R (>= 4.2)

Imports stats, utils

Suggests PhysioCore, PhysioAnnotationHub, S4Vectors, SummarizedExperiment, testthat (>= 3.2.0), GGIR, GGIRread, knitr, rmarkdown

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

Config/testthat/edition 3

URL <https://github.com/x-biosignal/PhysioWearable>,
<https://x-biosignal.r-universe.dev/PhysioWearable>,
<https://x-biosignal.github.io/PhysioWearable/>

BugReports <https://github.com/x-biosignal/PhysioWearable/issues>

Repository <https://x-biosignal.r-universe.dev>

Date/Publication 2026-10-03 13:11:45 UTC
RemoteUrl https://github.com/x-biosignal/PhysioWearable
RemoteRef HEAD
RemoteSha 2cb789716ed269b1366025ecf9c172e92c516f73

Contents

accelToPhysioExperiment	2
activityFragmentation	3
adlBudget	4
adlFeatures	4
adlToICF	5
autoCalibrateAccel	6
classifyBouts	7
coleKripke	8
computeENMO	9
computeENMOa	10
computeMAD	11
detectNonWear	12
detectSteps	13
diurnalProfile	14
freeLivingICF	14
intensityGradient	15
mxMetrics	16
paIntensityThresholds	17
readAccelCSV	17
recognizeADL	18
spo2Metrics	19
summarizeAppleSleep	20
summarizeFreeLiving	20
summarizeSleep	22
summarizeSleepStages	23
trainADL	24
upperLimbUse	25
Index	27

accelToPhysioExperiment
<i>Wrap tri-axial acceleration in a PhysioExperiment</i>

Description

Builds a PhysioExperiment (from PhysioCore) with a single 3-channel acceleration assay (time x c(x, y, z)) at the given sampling rate.

Usage

```
accelToPhysioExperiment(accel, sampling_rate, unit = "g", metadata = list())
```

Arguments

accel	An n x 3 matrix of acceleration (columns x, y, z), in g units.
sampling_rate	Sampling rate in Hz.
unit	Acceleration unit recorded in the channel metadata (default "g").
metadata	Optional named list stored in the object metadata.

Value

A PhysioExperiment object.

See Also

[readAccelCSV\(\)](#), [computeENMO\(\)](#)

activityFragmentation *Physical-activity fragmentation*

Description

Fragmentation metrics describe how activity and rest are *accumulated* – many short bouts (fragmented) versus few long ones. Reported for both the active and sedentary states: mean bout duration, bout counts, the transition probabilities (active-to-sedentary astp, sedentary-to-active satp; each the reciprocal of the mean bout length in the origin state, Chastin et al. 2010) and the Gini index of sedentary-bout durations.

Usage

```
activityFragmentation(intensity, epoch_sec)
```

Arguments

intensity	A per-epoch intensity factor (from classifyBouts()) or a logical/0-1 vector marking active epochs (TRUE = active). When a factor is given, any level other than "sedentary" counts as active.
epoch_sec	Epoch length in seconds.

Value

A list of fragmentation metrics (durations in minutes).

References

Chastin SFM, Granat MH (2010). Gait Posture 31(1):82-86.

See Also

[summarizeFreeLiving\(\)](#)

Examples

```
# alternating 4-epoch active / 6-epoch sedentary blocks
act <- rep(rep(c(TRUE, FALSE), c(4, 6)), 5)
activityFragmentation(act, epoch_sec = 60)$astp # 1/4 = 0.25
```

adlBudget	<i>Time budget of recognised activities</i>
-----------	---

Description

Time budget of recognised activities

Usage

```
adlBudget(recognized)
```

Arguments

recognized a [recognizeADL\(\)](#) result (or a data.frame with an activity column and start_sec).

Value

a data.frame activity, n_windows, minutes, proportion, ordered by time.

See Also

[adlToICF\(\)](#)

adlFeatures	<i>Windowed accelerometer features for activity recognition</i>
-------------	---

Description

Splits a raw tri-axial signal into fixed overlapping windows and computes, per window, the compact time- and frequency-domain feature set used for human activity recognition (per-axis and vector-magnitude mean/sd/MAD/energy/IQR/ range, the signal-magnitude area, inter-axis correlations, the dominant frequency, spectral energy and entropy, and jerk RMS). Dependency-free.

Usage

```
adlFeatures(accel, sampling_rate, window_sec = 2.56, overlap = 0.5)
```

Arguments

<code>accel</code>	An $n \times 3$ acceleration matrix (g units; columns x, y, z).
<code>sampling_rate</code>	Sampling rate in Hz.
<code>window_sec</code>	Window length in seconds (default 2.56, the UCI-HAR value).
<code>overlap</code>	Window overlap fraction in $[0, 1)$ (default 0.5).

Value

A data.frame with `window`, `start_sec`, `end_sec` and one column per feature (one row per window).

References

Anguita D, et al. (2013). ESANN.

See Also

[trainADL\(\)](#), [recognizeADL\(\)](#)

Examples

```
fs <- 20; t <- seq(0, 20, 1 / fs)
walk <- cbind(0.1 * sin(2 * pi * 1.8 * t), 0.1 * cos(2 * pi * 1.8 * t), 1)
head(adlFeatures(walk, fs), 3)
```

adlToICF

Map recognised ADL activities to ICF Activities & Participation codes

Description

Turns recognised activities into their ICF d-codes, stamped with the performance qualifier (real-world doing), and attaches the time budget when available. Activities are matched case-insensitively; unmapped labels are dropped with a warning. The result feeds the cross-modal ICF construct in *PhysioRehab*.

Usage

```
adlToICF(x)
```

Arguments

`x` a [recognizeADL\(\)](#) result, an [adlBudget\(\)](#) result, or a character vector of activity labels.

Value

a data.frame activity, `icf_code`, `icf_title`, `qualifier` ("performance") and, when a budget is available, `minutes` and `proportion` aggregated per ICF code.

See Also

[recognizeADL\(\)](#), [adlBudget\(\)](#), [freeLivingICF\(\)](#)

Examples

```
adlToICF(c("walking", "sitting", "walking", "eating"))
```

autoCalibrateAccel	<i>Gravity-based accelerometer auto-calibration</i>
--------------------	---

Description

Estimates a per-axis gain and offset that force the acceleration magnitude to 1 g during still periods, following the GGIR sphere-fitting method (van Hees et al. 2014). Still (non-movement) windows are identified by a low per-axis standard deviation; their mean orientations lie on a unit sphere in an ideally-calibrated device, and the gain/offset are fitted by iteratively projecting onto that sphere. Correction is $(\text{accel} - \text{offset}) * \text{scale}$.

Usage

```
autoCalibrateAccel(
  accel,
  sampling_rate,
  window_sec = 10,
  still_sd = 0.013,
  min_still = 10,
  max_iter = 100L,
  tol = 1e-06
)
```

Arguments

<code>accel</code>	An $n \times 3$ matrix of acceleration in g units.
<code>sampling_rate</code>	Sampling rate in Hz.
<code>window_sec</code>	Still-window length in seconds (default 10).
<code>still_sd</code>	Per-axis standard-deviation threshold for a still window (default 0.013 g).
<code>min_still</code>	Minimum number of still windows required (default 10).
<code>max_iter, tol</code>	Iteration cap and convergence tolerance.

Value

An `accel_calibration` list with `scale`, `offset`, `residual` (RMS of the still-point magnitude error after calibration), `n_still`, `iterations`, `converged`, and a `calibrate()` function that applies the correction to an $n \times 3$ matrix.

References

van Hees VT, et al. (2014). J Appl Physiol 117(7):738-744.

See Also

[computeENMO\(\)](#)

classifyBouts

Classify physical-activity intensity and detect activity bouts

Description

Classifies each ENMO epoch into a physical-activity intensity (sedentary, light, moderate, vigorous) and detects contiguous bouts at or above a target intensity lasting at least a minimum duration.

Usage

```
classifyBouts(
  enmo,
  epoch_sec,
  thresholds = paIntensityThresholds(),
  enmo_unit = c("mg", "g"),
  bout_level = c("moderate", "light", "vigorous"),
  min_bout_min = 1
)
```

Arguments

enmo	Per-epoch ENMO values (from computeENMO()).
epoch_sec	Epoch length in seconds.
thresholds	Named intensity thresholds in mg (see paIntensityThresholds()).
enmo_unit	Unit of enmo: "mg" (default) or "g".
bout_level	Minimum intensity for a bout: "light", "moderate" (default) or "vigorous".
min_bout_min	Minimum bout duration in minutes (default 1).

Value

A wearable_bouts list with intensity (per-epoch factor), a bouts data frame (start_epoch, end_epoch, n_epochs, duration_sec, mean_enmo), and minutes per intensity.

References

Hildebrand M, et al. (2014).

See Also

[computeENMO\(\)](#), [paIntensityThresholds\(\)](#)

 coleKripke

Cole-Kripke actigraphy sleep/wake scoring

Description

Scores each epoch as sleep or wake from per-epoch activity counts using the Cole-Kripke algorithm: a weighted sum of the surrounding epochs' activity is thresholded ($D < 1 \Rightarrow$ sleep). Weights are the classic 1-minute set.

Usage

```
coleKripke(counts, rescore = TRUE)
```

Arguments

counts	Numeric vector of per-epoch activity counts (one per minute).
rescore	If TRUE (default) apply the Webster rescoring rules that flip short isolated sleep bouts after sustained wake to wake.

Value

An integer vector the same length as counts: 1 = sleep, 0 = wake.

References

Cole RJ, Kripke DF, Gruen W, Mullaney DJ, Gillin JC (1992). "Automatic sleep/wake identification from wrist activity." Sleep 15:461-469.

See Also

[summarizeSleep\(\)](#), [summarizeAppleSleep\(\)](#)

Examples

```
set.seed(1)
counts <- c(rpois(30, 40), rpois(60, 2), rpois(20, 45)) # active, quiet, active
table(coleKripke(counts))
```

computeENMO	<i>Euclidean Norm Minus One (ENMO) acceleration metric</i>
-------------	--

Description

ENMO is the vector magnitude of tri-axial acceleration minus one gravity, with negative values truncated to zero (van Hees et al. 2013). For a perfectly still sensor (magnitude 1 g) ENMO is 0; movement raises it. When an epoch length is given, per-epoch mean ENMO is returned.

Usage

```
computeENMO(accel, sampling_rate = NULL, epoch_sec = NULL, unit = c("g", "mg"))
```

Arguments

accel	An n x 3 matrix (or data frame) of acceleration in g units (columns x, y, z).
sampling_rate	Sampling rate in Hz (required if epoch_sec is set).
epoch_sec	Epoch length in seconds for aggregation; NULL returns the per-sample ENMO.
unit	"g" (default) or "mg" (milli-g).

Value

A numeric vector of ENMO values (per sample, or per epoch).

References

van Hees VT, et al. (2013). PLoS ONE 8(4):e61691.

See Also

[classifyBouts\(\)](#), [autoCalibrateAccel\(\)](#)

Examples

```
still <- matrix(rep(c(0, 0, 1), each = 100), ncol = 3)
max(computeENMO(still))      # ~ 0 for a static 1 g signal
```

computeENMOa	<i>ENMO-abs (ENMOa) acceleration metric</i>
--------------	---

Description

ENMOa is the "always-positive" companion of `computeENMO`: the vector magnitude minus one gravity, with the *absolute value* taken instead of truncating negatives to zero, $ENMOa = |r - 1|$ with $r = \sqrt{x^2 + y^2 + z^2}$ (van Hees et al. 2013). It differs from ENMO only where the magnitude dips below 1 g (deceleration / partial free-fall), which ENMO truncates away; ENMOa retains that movement, so it is used when the sensor calibration is imperfect (a persistent sub-1 g offset would otherwise be lost by ENMO's truncation). When an epoch length is given, per-epoch mean ENMOa is returned.

Usage

```
computeENMOa(
  accel,
  sampling_rate = NULL,
  epoch_sec = NULL,
  unit = c("g", "mg")
)
```

Arguments

<code>accel</code>	An n x 3 matrix (or data frame) of acceleration in g units (columns x, y, z).
<code>sampling_rate</code>	Sampling rate in Hz (required if <code>epoch_sec</code> is set).
<code>epoch_sec</code>	Epoch length in seconds for aggregation; NULL returns the per-sample ENMOa.
<code>unit</code>	"g" (default) or "mg" (milli-g).

Value

A numeric vector of ENMOa values (per sample, or per epoch).

References

van Hees VT, et al. (2013). PLoS ONE 8(4):e61691.

See Also

`computeENMO()`, `computeMAD()`

Examples

```
set.seed(1)
acc <- matrix(rnorm(300, c(0, 0, 1), 0.1), ncol = 3, byrow = TRUE)
mean(computeENMOa(acc))      # >= mean(computeENMO): retains sub-1 g dips
```

computeMAD

*Mean Amplitude Deviation (MAD) acceleration metric***Description**

MAD is the mean absolute deviation of the tri-axial acceleration vector magnitude about its (per-epoch) mean (Vaha-Ypya et al. 2015): for each epoch, $MAD = \frac{1}{n} \sum_i |r_i - \bar{r}|$, where $r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$ is the sample vector magnitude and $\bar{r}$ its epoch mean. It is a gravity-robust movement-intensity metric complementary to [computeENMO](#): MAD captures the amount of *variation* in the magnitude within an epoch (so it is insensitive to a static orientation offset), whereas ENMO captures its *elevation* above 1 g. When an epoch length is given, per-epoch MAD is returned; otherwise a single whole-signal MAD.

Usage

```
computeMAD(accel, sampling_rate = NULL, epoch_sec = NULL, unit = c("g", "mg"))
```

Arguments

accel	An n x 3 matrix (or data frame) of acceleration in g units (columns x, y, z).
sampling_rate	Sampling rate in Hz (required if epoch_sec is set).
epoch_sec	Epoch length in seconds for aggregation; NULL returns a single MAD over the whole signal.
unit	"g" (default) or "mg" (milli-g).

Value

A numeric vector of MAD values (one per epoch, or a single value).

References

Vaha-Ypya H, et al. (2015). Clin Physiol Funct Imaging 35(1):64-70.

See Also

[computeENMO\(\)](#), [classifyBouts\(\)](#)

Examples

```
set.seed(1)
acc <- matrix(rnorm(300, c(0, 0, 1), 0.1), ncol = 3, byrow = TRUE)
computeMAD(acc) # whole-signal MAD (g)
computeMAD(acc, sampling_rate = 100, epoch_sec = 1) # per-epoch MAD
```

detectNonWear

*Detect non-wear periods***Description**

Flags non-wear windows where the accelerometer is stationary: at least `min_axes` of the three axes have a low standard deviation and value range over a long window (van Hees et al. 2011).

Usage

```
detectNonWear(
  accel,
  sampling_rate,
  window_min = 60,
  step_min = NULL,
  sd_thresh = 0.013,
  range_thresh = 0.05,
  min_axes = 2L
)
```

Arguments

<code>accel</code>	An $n \times 3$ matrix of acceleration in g units.
<code>sampling_rate</code>	Sampling rate in Hz.
<code>window_min</code>	Window length in minutes (default 60).
<code>step_min</code>	Window step in minutes; NULL uses <code>window_min / 4</code> . A sliding (overlapping) window prevents a non-wear period from being diluted across a block boundary.
<code>sd_thresh</code>	Per-axis standard-deviation threshold (default 0.013 g).
<code>range_thresh</code>	Per-axis value-range threshold (default 0.050 g).
<code>min_axes</code>	Number of axes that must meet both criteria (default 2).

Value

A nonwear list with a per-window windows data frame (`window`, `start_time`, `end_time`, `nonwear`) and the `nonwear_fraction` (fraction of samples covered by any non-wear window).

References

van Hees VT, et al. (2011). PLoS ONE 6(7):e22922.

detectSteps

*Detect steps from tri-axial acceleration***Description**

Counts steps by peak-detecting the dynamic acceleration magnitude (vector magnitude minus gravity). Peaks must exceed a prominence threshold and be separated by at least the shortest plausible step interval ($1 / \text{max_step_hz}$), which suppresses double counts.

Usage

```
detectSteps(
  accel,
  sampling_rate,
  min_step_hz = 0.5,
  max_step_hz = 4,
  threshold = NULL,
  min_amplitude = 0.05
)
```

Arguments

accel An $n \times 3$ matrix of acceleration in g units, or a numeric vector already representing an acceleration magnitude.

sampling_rate Sampling rate in Hz.

min_step_hz, max_step_hz Plausible step-frequency range in Hz (defaults 0.5 and 4).

threshold Minimum peak height in the dynamic magnitude (g). NULL uses $\max(0.5 * \text{sd}(\text{dynamic magnitude}), \text{min_amplitude})$, the absolute floor stops sensor noise in a still recording from being counted as steps.

min_amplitude Absolute floor (g) for the automatic threshold (default 0.05).

Value

A `wearable_steps` list with `n_steps`, `step_index` (sample indices of detected steps), and `cadence_spm` (steps per minute).

References

Any peak-based pedometer (e.g. Zijlstra & Hof 2003).

See Also

[computeENMO\(\)](#)

Examples

```
fs <- 50; t <- seq(0, 10, 1 / fs)
walk <- cbind(0, 0, 1 + 0.4 * sin(2 * pi * 1.8 * t)) # 1.8 steps/s
detectSteps(walk, fs)$n_steps
```

diurnalProfile	<i>Diurnal activity profile</i>
----------------	---------------------------------

Description

Mean ENMO by hour of day (0-23), averaged over all supplied epochs – the rest-activity rhythm of free-living behaviour. Requires per-epoch timestamps.

Usage

```
diurnalProfile(enmo_mg, timestamps)
```

Arguments

enmo_mg	Per-epoch ENMO in milli-g.
timestamps	Per-epoch POSIXct timestamps (same length as enmo_mg).

Value

A list with hourly (length-24 mean ENMO by hour), peak_hour, and morning/afternoon/evening/night mean ENMO.

See Also

[summarizeFreeLiving\(\)](#)

freeLivingICF	<i>Map free-living metrics to ICF codes (performance qualifier)</i>
---------------	---

Description

Returns the ICF Activities & Participation codes for free-living physical-behaviour metrics, each labelled with the ICF performance qualifier – the distinguishing feature of free-living monitoring, which captures what a person actually does in daily life rather than the capacity a clinic test measures. When a [summarizeFreeLiving\(\)](#) result is supplied, each mapped metric's aggregate value is attached.

Usage

```
freeLivingICF(x = NULL, hub = NULL)
```

Arguments

x	A freelifving_summary (from summarizeFreeLiving()) or a character vector of metric ids; NULL (default) returns the full free-living map.
hub	Optional PhysioAnnotationHub object passed through to the ontology lookup.

Details

Uses the maintained ontology in the suggested package **PhysioAnnotationHub** when available (its free-living rows), falling back to a built-in table otherwise.

Value

A data frame with metric, icf_code, icf_title, qualifier (always "performance") and, for a summary input, value.

See Also

[summarizeFreeLiving\(\)](#)

Examples

```
freeLivingICF() # the free-living metric -> ICF map
freeLivingICF(c("mvpa_min", "steps_per_day"))
```

intensityGradient	<i>Rowlands intensity gradient</i>
-------------------	------------------------------------

Description

The intensity gradient summarises the *distribution* of physical-activity intensity across the whole day: the log-log slope of time accumulated against acceleration intensity (Rowlands et al. 2018). A less negative gradient means relatively more time at higher intensities. Unlike a single MVPA cut-point it uses the entire intensity range and is cut-point free.

Usage

```
intensityGradient(enmo_mg, epoch_sec, bin_width_mg = 25, max_mg = 4000)
```

Arguments

enmo_mg	Per-epoch ENMO in milli-g.
epoch_sec	Epoch length in seconds.
bin_width_mg	Intensity bin width in mg (default 25).
max_mg	Upper edge of the binned range in mg (default 4000).

Value

A list with gradient (slope), intercept, r_squared, and the bins data frame (mid_mg, minutes) used in the regression.

References

Rowlands AV, et al. (2018). Med Sci Sports Exerc 50(6):1323-1332.

See Also

`mxMetrics()`, `summarizeFreeLiving()`

Examples

```
# a power-law intensity distribution: time proportional to intensity^-2
mids <- seq(12.5, 2000, by = 25)
minutes <- 1e5 * mids^(-2)
enmo <- rep(mids, round(minutes * 2))      # 30 s epochs -> 2 epochs/min
round(intensityGradient(enmo, epoch_sec = 30)$gradient, 2)  # ~ -2
```

mxMetrics

MX physical-activity intensity metrics

Description

The MX metrics describe the same distribution from the active end: MX is the acceleration (mg) above which a person's most active X minutes are accumulated (Rowlands et al. 2018). M60 is the intensity of the most active hour; M120 the most active two hours. Higher MX = more intense peak activity.

Usage

```
mxMetrics(enmo_mg, epoch_sec, minutes = c(2, 5, 15, 30, 60, 120))
```

Arguments

enmo_mg	Per-epoch ENMO in milli-g.
epoch_sec	Epoch length in seconds.
minutes	Integer vector of X values (minutes); default c(2, 5, 15, 30, 60, 120).

Value

A named numeric vector Mxx = mg; NA where the recording is shorter than X minutes.

References

Rowlands AV, et al. (2018).

See Also

[intensityGradient\(\)](#), [summarizeFreeLiving\(\)](#)

Examples

```
# most-active 60 min all at >= 200 mg
enmo <- c(rep(300, 120), rep(20, 2000)) # 30 s epochs -> 120 = 60 min
mxMetrics(enmo, epoch_sec = 30, minutes = 60)
```

paIntensityThresholds *Physical-activity intensity ENMO cut-points*

Description

Returns the ENMO thresholds (mg) separating physical-activity intensities for adults (Hildebrand et al. 2014 style): the lower bound of light, moderate and vigorous activity.

Usage

```
paIntensityThresholds()
```

Value

A named numeric vector `c(light, moderate, vigorous)` in mg.

References

Hildebrand M, et al. (2014). Med Sci Sports Exerc 46(9):1816-1824.

See Also

[classifyBouts\(\)](#)

readAccelCSV *Read a raw accelerometer CSV export into a PhysioExperiment*

Description

Reads a tri-axial accelerometer CSV (GENEActiv / ActiGraph / Axivity exports and similar) and wraps it in a `PhysioExperiment`. The x/y/z columns are resolved by name (case-insensitive x/y/z, optionally with an `accel_` prefix) or by the `columns` argument.

Usage

```
readAccelCSV(path, sampling_rate, columns = NULL, skip = 0, unit = "g", ...)
```

Arguments

<code>path</code>	Path to the CSV file.
<code>sampling_rate</code>	Sampling rate in Hz.
<code>columns</code>	Optional length-3 character (column names) or integer (column indices) selecting the x, y, z columns; NULL auto-detects.
<code>skip</code>	Number of header lines to skip before the column header (default 0).
<code>unit</code>	Acceleration unit (default "g").
<code>...</code>	Further arguments passed to <code>utils::read.csv()</code> .

Value

A `PhysioExperiment` object.

See Also

[accelToPhysioExperiment\(\)](#)

recognizeADL

Recognise ADL activities in a raw recording

Description

Windows a raw tri-axial signal, extracts [adlFeatures\(\)](#) and classifies each window with a trained [trainADL\(\)](#) model, giving the activity per window.

Usage

```
recognizeADL(
  model,
  accel,
  sampling_rate,
  window_sec = model$window_sec,
  overlap = model$overlap
)
```

Arguments

<code>model</code>	an <code>adl_model</code> .
<code>accel</code>	an $n \times 3$ acceleration matrix (g units).
<code>sampling_rate</code>	sampling rate in Hz.
<code>window_sec, overlap</code>	window settings (default: the model's).

Value

a data.frame with `window`, `start_sec`, `end_sec` and `activity`.

See Also

[adlBudget\(\)](#), [adlToICF\(\)](#)

spo2Metrics	<i>Oximetry (SpO2) metrics</i>
-------------	--------------------------------

Description

Computes the standard blood-oxygen summary indices from an SpO2 series. Accepts SpO2 as a percentage (90) or a fraction (0.90; auto-detected and scaled). When time is supplied the time-below-threshold and desaturation index are duration-weighted; otherwise they are sample proportions/counts.

Usage

```
spo2Metrics(spo2, time = NULL, threshold = 90, desat = 3, window = 12)
```

Arguments

spo2	Numeric SpO2 series (percent or 0-1 fraction).
time	Optional sample times as POSIXct or numeric seconds (same length and order as spo2). Enables t90_min and per-hour odi.
threshold	Desaturation threshold percent for T90/CT90 (default 90).
desat	Drop (percentage points) below the rolling baseline that defines a desaturation event (default 3; 4 is also common).
window	Rolling-baseline length in samples for event detection (default 12).

Value

An spo2_metrics object: mean, nadir, pct_below, ct90, t90_min, n_desat, odi (events/hour), plus the settings.

References

Chung F et al. (2012) oximetry desaturation indices; standard ODI/CT90 definitions used in sleep-disordered-breathing screening.

See Also

[summarizeAppleSleep\(\)](#)

Examples

```
set.seed(1)
spo2 <- pmin(100, pmax(80, 97 + round(rnorm(200))))
spo2[50:55] <- 88 # a desaturation
spo2Metrics(spo2)
```

summarizeAppleSleep	<i>Summarise Apple Watch sleep stages into clinical metrics</i>
---------------------	---

Description

Apple-specific wrapper around [summarizeSleepStages\(\)](#) for the SleepAnalysis stage intervals the Apple Watch records (from `PhysioDevices::appleHealthSeries(x, "SleepAnalysis")`): asleep is AsleepCore/Deep/REM, and core_min/deep_min/rem_min are reported.

Usage

```
summarizeAppleSleep(stages, gap_min = 120)
```

Arguments

stages	A data frame with start, end (POSIXct) and value ("AsleepCore", "AsleepDeep", "AsleepREM", "Awake", "InBed").
gap_min	Minutes of gap that start a new night (default 120).

Value

See [summarizeSleepStages\(\)](#); with core_min/deep_min/rem_min.

See Also

[summarizeSleepStages\(\)](#), [summarizeSleep\(\)](#)

Examples

```
t0 <- as.POSIXct("2023-05-01 23:00:00", tz = "UTC")
stages <- data.frame(
  start = t0 + c(0, 3600, 3720, 7200),
  end   = t0 + c(3600, 3720, 7200, 10800),
  value = c("AsleepCore", "Awake", "AsleepDeep", "AsleepREM"))
summarizeAppleSleep(stages)
```

summarizeFreeLiving	<i>Summarise free-living physical behaviour over one or more days</i>
---------------------	---

Description

The umbrella summary: from multi-day per-epoch ENMO (plus optional timestamps and a wear mask) it computes, per valid day and averaged across valid days, the physical-behaviour metrics that free-living accelerometry studies report – intensity time-use (sedentary / light / MVPA), activity volume, the Rowlands intensity gradient and MX metrics, activity fragmentation, the diurnal profile and WHO guideline attainment. Only days with enough wear time are counted as valid; non-wear epochs are excluded from the intensity tallies.

Usage

```
summarizeFreeLiving(
  enmo,
  epoch_sec,
  timestamps = NULL,
  wear = NULL,
  enmo_unit = c("mg", "g"),
  thresholds = paIntensityThresholds(),
  valid_wear_hours = 10,
  steps_per_day = NULL,
  guideline_mvpa_min = 150
)
```

Arguments

<code>enmo</code>	Per-epoch ENMO (see computeENMO()).
<code>epoch_sec</code>	Epoch length in seconds.
<code>timestamps</code>	Optional per-epoch POSIXct timestamps; when supplied, days are split on the calendar date and the diurnal profile is computed.
<code>wear</code>	Optional per-epoch logical wear mask (TRUE = worn). Default: all worn.
<code>enmo_unit</code>	Unit of enmo: "mg" (default) or "g".
<code>thresholds</code>	Intensity cut-points in mg (see paIntensityThresholds()).
<code>valid_wear_hours</code>	Minimum wear hours for a day to be valid (default 10).
<code>steps_per_day</code>	Optional numeric vector of steps for each day (in day order) folded into the per-day table and averages.
<code>guideline_mvpa_min</code>	Weekly MVPA-equivalent minutes for guideline attainment (default 150; WHO 2020, counting vigorous double).

Value

A `freeliving_summary` object: `by_day` (per-day data frame), `aggregate` (means over valid days plus guideline attainment), `diurnal` (or NULL), `n_valid_days` and settings.

References

Rowlands (2018); Chastin (2010); WHO (2020).

See Also

[intensityGradient\(\)](#), [mxMetrics\(\)](#), [activityFragmentation\(\)](#), [freeLivingICF\(\)](#)

Examples

```
set.seed(1)
# one day, 30 s epochs: mostly sedentary with an active hour
enmo <- c(rep(5, 2 * 60 * 10), rep(120, 2 * 60), rep(5, 2 * 60 * 12))
s <- summarizeFreeLiving(enmo, epoch_sec = 30, valid_wear_hours = 0)
s$aggregate$mvpa_min
```

summarizeSleep	<i>Summarise a sleep/wake series into standard sleep metrics</i>
----------------	--

Description

Summarise a sleep/wake series into standard sleep metrics

Usage

```
summarizeSleep(sleep_wake, epoch = 60, spt = NULL)
```

Arguments

sleep_wake	Integer/logical per-epoch sleep(1)/wake(0) vector (e.g. from coleKripke()).
epoch	Epoch length in seconds (default 60).
spt	Optional integer length-2 vector c(start, end) of epoch indices bounding the sleep-period (time in bed). If NULL (default) it spans the first to the last sleep epoch.

Value

A one-row data frame: tib_min (time in bed), tst_min (total sleep time), sleep_efficiency (TST/TIB), waso_min (wake after sleep onset), sol_min (sleep-onset latency), n_awakenings.

See Also

[coleKripke\(\)](#)

Examples

```
sw <- c(rep(0, 5), rep(1, 40), 0, 0, rep(1, 30))
summarizeSleep(sw)
```

summarizeSleepStages *Summarise staged sleep into per-night clinical metrics*

Description

Vendor-agnostic engine behind [summarizeAppleSleep\(\)](#): given sleep-stage intervals labelled by any scheme (Apple AsleepCore/Deep/REM, Fitbit light/deep/rem/wake, ...), compute per-night sleep metrics. Consecutive intervals are grouped into nights whenever the gap between them exceeds `gap_min`.

Usage

```
summarizeSleepStages(
  stages,
  asleep_levels,
  wake_levels = "Awake",
  inbed_levels = "InBed",
  stage_cols = NULL,
  gap_min = 120
)
```

Arguments

<code>stages</code>	A data frame with <code>start</code> , <code>end</code> (POSIXct) and a stage-label column named <code>value</code> or <code>stage</code> .
<code>asleep_levels</code>	Labels counted as asleep (e.g. Apple <code>c("AsleepCore", "AsleepDeep", "AsleepREM")</code> ; Fitbit <code>c("light", "deep", "rem")</code>).
<code>wake_levels</code>	Labels counted as wake within the sleep period (default "Awake"; Fitbit "wake").
<code>inbed_levels</code>	Labels counted as explicit time in bed (default "InBed").
<code>stage_cols</code>	Optional named character vector mapping output columns to stage labels, e.g. <code>c(deep = "deep", rem = "rem")</code> adds <code>deep_min</code> , <code>rem_min</code> .
<code>gap_min</code>	Minutes of gap that start a new night (default 120).

Value

A data frame, one row per night: `night`, `start`, `end`, `tib_min`, `tst_min`, `sleep_efficiency`, `waso_min`, `sol_min`, `n_awakenings`, and one `<name>_min` column per `stage_cols` entry.

See Also

[summarizeAppleSleep\(\)](#), [summarizeSleep\(\)](#), [coleKripke\(\)](#)

Examples

```
t0 <- as.POSIXct("2023-05-01 23:00:00", tz = "UTC")
stages <- data.frame(start = t0 + c(0, 3600, 7200),
                     end = t0 + c(3600, 7200, 10800),
                     stage = c("light", "deep", "rem"))
summarizeSleepStages(stages, asleep_levels = c("light", "deep", "rem"),
                     wake_levels = "wake", stage_cols = c(deep = "deep", rem = "rem"))
```

trainADL

Train an ADL activity recogniser from labelled windows

Description

Learns a k-nearest-neighbours classifier over standardised [adlFeatures\(\)](#), mapping a window's features to an activity label. Dependency-free and model-agnostic: for a stronger engine, extract features here and classify with [PhysioML](#) (ROCKET/torch) instead – the ICF mapping ([adlToICF\(\)](#)) is unchanged.

Usage

```
trainADL(features, labels, k = 5, window_sec = 2.56, overlap = 0.5)
```

```
## S3 method for class 'adl_model'
predict(object, newdata, ...)
```

Arguments

features	Training features: an adlFeatures() data.frame or a numeric feature matrix (one row per window).
labels	Activity label per window (length = number of windows).
k	Neighbours for the vote (default 5).
window_sec, overlap	Window settings to store so recognizeADL() can re-extract features consistently (defaults match adlFeatures()).
object	an adl_model.
newdata	features to classify (adlFeatures() data.frame or matrix).
...	unused.

Value

an adl_model object.

See Also

[recognizeADL\(\)](#), [adlFeatures\(\)](#), [adlToICF\(\)](#)

Examples

```

set.seed(1)
fs <- 20; t <- seq(0, 30, 1 / fs)
walk <- cbind(0.2 * sin(2 * pi * 1.8 * t), 0.2 * cos(2 * pi * 1.8 * t), 1)
sit <- cbind(rnorm(length(t), 0, 0.01), rnorm(length(t), 0, 0.01), 1)
fw <- adlFeatures(walk, fs); fs2 <- adlFeatures(sit, fs)
feats <- rbind(fw, fs2)
labs <- c(rep("walking", nrow(fw)), rep("sitting", nrow(fs2)))
m <- trainADL(feats, labs)
m

```

upperLimbUse

*Real-world upper-limb use from bilateral wrist accelerometry***Description**

The field-standard free-living performance metrics for arm use – the sensor counterpart to a clinic upper-limb capacity test. From the affected (or dominant) and unaffected (or non-dominant) wrist activity it computes the Use Ratio, the Magnitude Ratio, the Bilateral Magnitude, the hours of use per arm and a laterality index. Maps to ICF d445 (hand and arm use), the performance substrate of the d5 self-care activities.

Usage

```

upperLimbUse(
  affected,
  unaffected,
  sampling_rate = NULL,
  epoch_sec = 5,
  active_threshold = 2
)

```

Arguments

affected, unaffected	The two arms' data: each an n x 3 acceleration matrix (g units) or a numeric vector of per-epoch ENMO (mg).
sampling_rate	Sampling rate in Hz (required for raw accel input).
epoch_sec	Epoch length in seconds (default 5).
active_threshold	ENMO (mg) above which an epoch counts as active (default 2).

Value

an upper_limb_use list: icf_code, use_ratio (affected/unaffected active time; 1 = symmetric, <1 = affected used less), magnitude_ratio (median log activity ratio, capped +/-7; 0 = symmetric, negative = unaffected dominant), bilateral_magnitude (median combined activity, mg), hours_affected/hours_unaffected, and laterality_index ((aff-unaff)/(aff+unaff)).

References

Uswatte (2005); Bailey (2014); Lang (2017).

See Also

[summarizeFreeLiving\(\)](#), [recognizeADL\(\)](#)

Examples

```
# affected arm active half as often as the unaffected arm (per-epoch ENMO, mg)
set.seed(1)
unaff <- c(rep(10, 100), rep(0.5, 100))
aff    <- c(rep(10, 50),  rep(0.5, 150))
upperLimbUse(aff, unaff)$use_ratio      # ~ 0.5
```

Index

accelToPhysioExperiment, [2](#)
accelToPhysioExperiment(), [18](#)
activityFragmentation, [3](#)
activityFragmentation(), [21](#)
adlBudget, [4](#)
adlBudget(), [5](#), [6](#), [19](#)
adlFeatures, [4](#)
adlFeatures(), [18](#), [24](#)
adlToICF, [5](#)
adlToICF(), [4](#), [19](#), [24](#)
autoCalibrateAccel, [6](#)
autoCalibrateAccel(), [9](#)

classifyBouts, [7](#)
classifyBouts(), [3](#), [9](#), [11](#), [17](#)
coleKripke, [8](#)
coleKripke(), [22](#), [23](#)
computeENMO, [9](#), [10](#), [11](#)
computeENMO(), [3](#), [7](#), [10](#), [11](#), [13](#), [21](#)
computeENMOa, [10](#)
computeMAD, [11](#)
computeMAD(), [10](#)

detectNonWear, [12](#)
detectSteps, [13](#)
diurnalProfile, [14](#)

freeLivingICF, [14](#)
freeLivingICF(), [6](#), [21](#)

intensityGradient, [15](#)
intensityGradient(), [17](#), [21](#)

mxMetrics, [16](#)
mxMetrics(), [16](#), [21](#)

paIntensityThresholds, [17](#)
paIntensityThresholds(), [7](#), [21](#)
predict.adl_model(trainADL), [24](#)

readAccelCSV, [17](#)

readAccelCSV(), [3](#)
recognizeADL, [18](#)
recognizeADL(), [4–6](#), [24](#), [26](#)

spo2Metrics, [19](#)
summarizeAppleSleep, [20](#)
summarizeAppleSleep(), [8](#), [19](#), [23](#)
summarizeFreeLiving, [20](#)
summarizeFreeLiving(), [4](#), [14–17](#), [26](#)
summarizeSleep, [22](#)
summarizeSleep(), [8](#), [20](#), [23](#)
summarizeSleepStages, [23](#)
summarizeSleepStages(), [20](#)

trainADL, [24](#)
trainADL(), [5](#), [18](#)

upperLimbUse, [25](#)
utils::read.csv(), [18](#)