Package: mrpheus (via r-universe)

November 2, 2026

Title Raw Physiological Signal Analysis for Biological Rhythms Research

Version 0.1.4

Description Raw physiological signal analysis for biological rhythms research. Provides ingestion, artefact detection, event detection, and feature extraction for multi-modal physiological recordings including polysomnography (EDF/EDF+), MRI-concurrent physiological logs (Philips PMU), and EEG. Covers cardiac rhythm (QRS detection, HRV), respiratory rhythm (apnoea detection, respiratory indices), and neural oscillations (sleep spindles, slow oscillations, automatic AASM sleep staging via a pre-trained LightGBM model ported from YASA; Vallat & Walker, 2021). Exports staged hypnograms to hypnor and derived metrics to syncR. Part of the Circadia Lab R ecosystem at Northumbria University.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

URL https://mrpheus.circadia-lab.uk, https://github.com/circadia-bio/mrpheus

BugReports https://github.com/circadia-bio/mrpheus/issues

Depends R (>= 4.1.0)

Imports fastICA (>= 1.2-0), edfReader (>= 1.2.1), gsignal (>= 0.3.5), pracma (>= 2.4.2), lightgbm (>= 4.0.0), dplyr (>= 1.1.0), tibble (>= 3.2.0), cli (>= 3.6.0), rlang (>= 1.1.0), Rcpp (>= 1.0.10)

LinkingTo Rcpp

Suggests scales, testthat (>= 3.0.0), knitr, rmarkdown, withr, ggplot2 (>= 3.4.0)

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Config/Needs/website tidyr, ggplot2, scales

Repository https://circadia-bio.r-universe.dev

Date/Publication 2026-07-08 22:16:43 UTC

RemoteUrl https://github.com/circadia-bio/mrpheus

RemoteRef v0.1.4

RemoteSha eac294b569229c02144e5186599d369edcad3c07

Contents

bandpass_filter . . . 3
compute_ahi . . . 4
compute_band_power . . . 4
compute_hr_signal . . . 6
compute_hrv_sleep . . . 7
compute_odi . . . 8
compute_slow_oscillations . . . 8
compute_spectrogram . . . 10
compute_spindles . . . 11
compute_temporal_bandpower . . . 12
correct_eog_ica . . . 14
correct_eog_regression . . . 15
detect_apneas . . . 16
detect_artifacts . . . 17
detect_powerline . . . 18
detect_qrs . . . 19
export_hypnogram . . . 21
notch_filter . . . 22
palette_orpheus . . . 23
prepare_psg . . . 24
preprocess_psg . . . 25
read_edf . . . 27
read_philips_physlog . . . 28
remove_dc . . . 29
stage_epochs . . . 30

Index **32**

`bandpass_filter` *Apply a bandpass filter to a signal*

Description

Designs a 4th-order Butterworth bandpass filter and applies it zero-phase using `gsignal::filtfilt()`. Handles edge cases gracefully:

Usage

```
bandpass_filter(signal, sr, low_hz, high_hz)
```

Arguments

`signal`	Numeric vector.
`sr`	Numeric. Sampling rate in Hz.
`low_hz`	Numeric. Low cutoff frequency in Hz. Use `0` or negative to apply a low-pass filter only.
`high_hz`	Numeric. High cutoff frequency in Hz.

Details

- `low_hz <= 0` → low-pass at `high_hz`
- `high_hz >= Nyquist` → high-pass at `low_hz`
- Otherwise → bandpass between `low_hz` and `high_hz`

The `high_hz` cutoff is clamped to 99 % of Nyquist to avoid instability. Because this uses a short IIR filter, it is safe on long recordings.

Value

Filtered numeric vector, same length as `signal`.

Examples

```
## Not run:
# Standard EEG bandpass
filtered <- bandpass_filter(eeg_signal, sr = 256, low_hz = 0.3, high_hz = 35)

# High-pass only (baseline drift removal)
hp <- bandpass_filter(eeg_signal, sr = 256, low_hz = 0.1, high_hz = Inf)

## End(Not run)
```

compute_ahi *Compute Apnea-Hypopnea Index (AHI)*

Description

Calculates the AHI from respiratory event data and total sleep time.

Usage

```
compute_ahi(events, tst_hours)
```

Arguments

events	Output of detect_apneas().
tst_hours	Numeric. Total sleep time in hours.

Value

Numeric scalar. AHI (events per hour).

compute_band_power *Compute EEG band power per epoch*

Description

Estimates power spectral density (PSD) using Welch's method and integrates power within standard EEG frequency bands for each epoch and channel.

Usage

```
compute_band_power(
  psg,
  channels = NULL,
  bands = list(delta = c(0.5, 4), theta = c(4, 8), alpha = c(8, 12), sigma = c(12, 16),
    beta = c(16, 30), gamma = c(30, 45)),
  relative = FALSE,
  win_sec = 4,
  noverlap = 200L,
  nfft = 1024L
)
```

Arguments

`psg`	An `mrpheus_psg` object from `prepare_psg()`.
`channels`	Character vector. Channel labels to include. If `NULL` (default), all non-bad EEG channels are used.
`bands`	Named list of length-2 numeric vectors defining frequency bands in Hz. Default matches YASA's `bandpower()`: `list(delta = c(0.5, 4), theta = c(4, 8), alpha = c(8, 12), sigma = c(12, 16), beta = c(16, 30), gamma = c(30, 45))`
`relative`	Logical. If `TRUE`, each band power is divided by the sum of all band powers (dimensionless). Matches YASA `relative = True`. Default `FALSE`.
`win_sec`	Numeric. Welch window length in seconds. Default `4`.
`noverlap`	Integer. Number of overlapping samples between Welch windows. Default `200`. Must be less than `win_sec * sample_rate`.
`nfft`	Integer. FFT length. Default `1024`. If smaller than the window length in samples it is automatically raised to the next power of 2.

Details

The Welch implementation matches `scipy.signal.welch` with YASA's default parameters (`win_sec = 4`, `noverlap = 200`, `nfft = 1024`, Hann window, constant detrend, mean averaging). Relative power matches YASA's `relative = True` behaviour: each band is divided by the sum of all band powers (not the total PSD integral).

Value

A tibble with columns `epoch`, `channel`, one column per named band, and `total_power` (sum of all band powers, in V^2/Hz units before any relative scaling).

Examples

```
## Not run:
bp <- compute_band_power(psg)
bp <- compute_band_power(psg, channels = "EEG Fpz-Cz", relative = TRUE)

# Custom bands
bp <- compute_band_power(
  psg,
  bands = list(slow = c(0.5, 1), delta = c(1, 4), theta = c(4, 8))
)

## End(Not run)
```

`compute_hr_signal` *Compute Instantaneous Heart Rate from R-Peak Indices*

Description

Converts R-peak sample indices (from `detect_qrs()`) into a sample-by-sample instantaneous heart rate trace by filling each inter-beat interval with the constant HR derived from that interval's length.

Usage

```
compute_hr_signal(qrs)
```

Arguments

`qrs`	A `mrpheus_qrs` object from `detect_qrs()`, or a named list with at least `qrs_i` (integer vector of R-peak indices) and `fs` (sampling frequency in Hz).

Value

Numeric vector of length `max(qrs$qrs_i)`. Each sample contains the instantaneous heart rate in beats per minute for the inter-beat interval it falls within. Samples before the first detected R-peak are 0.

See Also

`detect_qrs()`, `compute_hrv_sleep()`

Examples

```
## Not run:
rec <- read_philips_physlog("sub-01_physlog.log")
qrs <- detect_qrs(rec$C[, "v1raw"], fs = rec$HDR$sfreq)
hr  <- compute_hr_signal(qrs)

plot(seq_along(hr) / rec$HDR$sfreq, hr, type = "l",
     xlab = "Time (s)", ylab = "HR (bpm)")

## End(Not run)
```

`compute_hrv_sleep` *Compute HRV metrics across sleep stages*

Description

Extracts R-peak positions from the ECG channel, computes standard time- and frequency-domain heart rate variability (HRV) metrics, and stratifies results by sleep stage.

Usage

```
compute_hrv_sleep(
  psg,
  staging = NULL,
  ecg_channel = NULL,
  min_rr_ms = 300,
  max_rr_ms = 2000
)
```

Arguments

`psg`	An `mrpheus_psg` object from `prepare_psg()`.
`staging`	Tibble from `stage_epochs()` or `NULL`. If `NULL`, HRV is computed across all epochs without stage stratification.
`ecg_channel`	Character or `NULL`. ECG channel label. If `NULL` (default), the first non-bad ECG channel is used.
`min_rr_ms`	Numeric. Minimum physiologically plausible RR interval (ms). Default `300` (200 bpm ceiling).
`max_rr_ms`	Numeric. Maximum physiologically plausible RR interval (ms). Default `2000` (30 bpm floor).

Value

A tibble with one row per sleep stage (or one row if `staging` is `NULL`):

stage Character. AASM stage or `"ALL"`.

n_epochs Integer. Number of epochs in this stage.

mean_rr_ms Numeric. Mean RR interval (ms).

sdnn_ms Numeric. SDNN — SD of all NN intervals.

rmssd_ms Numeric. RMSSD — root mean square of successive differences.

lf_power Numeric. LF band power (0.04–0.15 Hz).

hf_power Numeric. HF band power (0.15–0.4 Hz).

lf_hf_ratio Numeric. LF/HF ratio.

compute_odi *Compute Oxygen Desaturation Index (ODI)*

Description

Calculates the ODI (number of >= 3% SpO2 desaturations per hour of sleep) from the SpO2 channel.

Usage

```
compute_odi(psg, spo2_channel, tst_hours, threshold = 3)
```

Arguments

psg	An mrpheus_psg object from prepare_psg().
spo2_channel	Character. SpO2 channel label.
tst_hours	Numeric. Total sleep time in hours.
threshold	Numeric. Desaturation threshold (percentage points). Default 3.

Value

Numeric scalar. ODI (desaturations per hour).

compute_slow_oscillations *Detect slow oscillations*

Description

Detects slow oscillations (SOs) in EEG channels using a zero-crossing approach in the delta band (0.5-2 Hz), following the algorithm described in Molle et al. (2002) and implemented in YASA (Vallat & Walker, 2021).

Usage

```
compute_slow_oscillations(
  psg,
  channel = NULL,
  stages = NULL,
  freq_so = c(0.5, 2),
  amp_ptp_threshold_uv = c(75, 500),
  duration_pos_s = c(0.1, 1),
  duration_neg_s = c(0.1, 1.5)
)
```

Arguments

`psg`	An `mrpheus_psg` object from `prepare_psg()`.
`channel`	Character. EEG channel label. If `NULL` (default), the first non-bad EEG channel is used.
`stages`	Integer vector or `NULL`. Epoch indices restricted to N2/N3. If `NULL`, detection runs across all epochs.
`freq_so`	Numeric vector of length 2. SO frequency band (Hz). Default `c(0.5, 2)`.
`amp_ptp_threshold_uv`	Numeric vector of length 2. Min and max acceptable peak-to-peak amplitude (uV). Default `c(75, 500)`.
`duration_pos_s`	Numeric vector of length 2. Min and max positive half- wave duration (s). Default `c(0.1, 1.0)`.
`duration_neg_s`	Numeric vector of length 2. Min and max negative half- wave duration (s). Default `c(0.1, 1.5)`.

Value

A tibble with one row per detected slow oscillation:

epoch Integer.

start_s Numeric. Onset (s) relative to epoch start.

end_s Numeric. Offset (s) relative to epoch start.

duration_s Numeric.

neg_peak_uv Numeric. Negative peak amplitude (uV).

pos_peak_uv Numeric. Positive peak amplitude (uV).

ptp_uv Numeric. Peak-to-peak amplitude (uV).

channel Character.

References

Molle, M., Marshall, L., Gais, S., & Born, J. (2002). Grouping of spindle activity during slow oscillations in human non-rapid eye movement sleep. *Journal of Neuroscience*, 22(24), 10941-10947.

Vallat, R., & Walker, M. P. (2021). An open-source, high-performance tool for automated sleep staging. *eLife*, 10, e70092. doi:10.7554/eLife.70092

`compute_spectrogram` *Compute a time-frequency spectrogram*

Description

Generates a short-time Fourier transform (STFT) based spectrogram for a single PSG channel across all epochs. Returns power ($\mu V^2/Hz$) as a matrix with time (epochs) on rows and frequency on columns.

Usage

```
compute_spectrogram(
  psg,
  channel = NULL,
  freq_range = c(0, 40),
  window_s = 2,
  overlap = 0.5,
  db = TRUE
)
```

Arguments

`psg`	An `mrpheus_psg` object from `prepare_psg()`.
`channel`	Character. A single channel label.
`freq_range`	Numeric vector of length 2. Frequency range to return (Hz). Default `c(0, 40)`.
`window_s`	Numeric. STFT window length in seconds. Default `2`.
`overlap`	Numeric in [0, 1). Fractional window overlap. Default `0.5`.
`db`	Logical. Return power in decibels (`10 * log10(power)`)? Default `TRUE`.

Value

A list of class `mrpheus_spectrogram` with:

power Numeric matrix. Rows = epochs, columns = frequency bins.

freqs Numeric vector. Frequency bin centres (Hz).

epochs Integer vector. Epoch indices.

channel Character. The channel label.

db Logical. Whether power is in dB.

`compute_spindles` *Detect sleep spindles*

Description

Detects sleep spindles in EEG channels using a band-pass / RMS envelope approach, closely following the algorithm in Lacourse et al. (2019) and implemented in YASA (Vallat & Walker, 2021). Spindles are identified as transient bursts of 11-16 Hz activity during NREM sleep.

Usage

```
compute_spindles(
  psg,
  channel = NULL,
  stages = NULL,
  freq_spindle = c(11, 16),
  rms_window_s = 0.3,
  min_duration_s = 0.5,
  max_duration_s = 3,
  threshold_sd = 1.5
)
```

Arguments

`psg`	An `mrpheus_psg` object from `prepare_psg()`.
`channel`	Character. EEG channel label. If `NULL` (default), the first non-bad EEG channel is used.
`stages`	Integer vector or `NULL`. Epoch indices (1-based) restricted to NREM sleep. If `NULL`, spindles are detected across all epochs.
`freq_spindle`	Numeric vector of length 2. Spindle frequency band (Hz). Default `c(11, 16)`.
`rms_window_s`	Numeric. Moving RMS window length in seconds. Default `0.3`.
`min_duration_s`	Numeric. Minimum spindle duration in seconds. Default `0.5`.
`max_duration_s`	Numeric. Maximum spindle duration in seconds. Default `3.0`.
`threshold_sd`	Numeric. RMS threshold as a multiple of the channel SD (within sigma band). Default `1.5`.

Value

A tibble with one row per detected spindle:

epoch Integer. Epoch in which the spindle was detected.

start_s Numeric. Spindle onset relative to epoch start (seconds).

end_s Numeric. Spindle offset relative to epoch start (seconds).

duration_s Numeric. Spindle duration in seconds.

peak_freq_hz Numeric. Peak frequency within the spindle.

rms_uv Numeric. Mean RMS amplitude within the spindle (uV).

channel Character. Channel label.

References

Lacourse, K., Yetton, B., Mednick, S., & Bhatt, D. L. (2019). *Massive online sleep staging: A polysomnography data repository.* Journal of Sleep Research.

Vallat, R., & Walker, M. P. (2021). An open-source, high-performance tool for automated sleep staging. *eLife*, 10, e70092. doi:10.7554/eLife.70092

`compute_temporal_bandpower`

Compute temporal band power across a PSG recording

Description

Applies a sliding window over the scored epochs, concatenates the signal within each window, estimates the Welch PSD averaged across channels, and integrates power within standard EEG frequency bands. Returns a long-format tibble suitable for plotting temporal power dynamics across the night.

Usage

```
compute_temporal_bandpower(
  psg,
  hypno,
  channels = NULL,
  window_epochs = 10L,
  step_epochs = NULL,
  bands = list(delta = c(0.5, 4), theta = c(4, 8), alpha = c(8, 12), sigma = c(12, 16),
    beta = c(16, 30), gamma = c(30, 45)),
  win_sec = 4,
  noverlap = NULL,
  nfft = NULL
)
```

Arguments

`psg`	An `mrpheus_psg` object from `prepare_psg()`.
`hypno`	Integer vector of length `psg$n_epochs`. Sleep stage codes following the YASA convention: `0` = Wake, `1` = N1, `2` = N2, `3` = N3, `4` = REM, `-1` = Artefact. Used to assign a dominant stage to each window.

`channels` Character vector. Channel labels whose PSDs are averaged. If `NULL` (default), all non-bad EEG channels are used.

`window_epochs` Integer. Number of 30-s epochs per window. Default `10` (5 minutes).

`step_epochs` Integer. Window step size in epochs. Default `NULL`, which sets the step to `window_epochs %/% 2` (50 % overlap).

`bands` Named list of length-2 numeric vectors (Hz). Default matches `compute_band_power()`:

```
list(delta = c(0.5, 4), theta = c(4, 8), alpha = c(8, 12),
     sigma = c(12, 16), beta = c(16, 30), gamma = c(30, 45))
```

`win_sec` Numeric. Welch window length in seconds. Default `4`.

`noverlap` Integer or `NULL`. Welch overlap in samples. `NULL` (default) sets it to `win_sec / 2 * sample_rate`, i.e. 50 % of the Welch window — matching the notebook's `noverlap = int(2 * sfreq)`.

`nfft` Integer or `NULL`. FFT length. `NULL` (default) uses the next power of 2 above `win_sec * sample_rate`.

Details

Mirrors `calculate_temporal_bandpower()` from the companion Python notebook: windows step by 50 % of their length (default), Welch overlap defaults to half the window length in samples (proportional to sample rate), and relative power is the share of each band in the total band-summed power.

Value

A tibble with columns:

time_hours Numeric. Start time of the window in hours from recording onset.

epoch_start Integer. First epoch index in the window (1-based).

epoch_end Integer. Last epoch index in the window (1-based).

dominant_stage Integer. Most common stage code across the window (artefact epochs excluded). `NA` if all epochs are artefacts.

band Character. Band name.

power Numeric. Mean band power across channels (V^2/Hz).

relative_power Numeric. Band power as a fraction of total band-summed power at this time point.

Examples

```
## Not run:
rec   <- read_edf("SC4001E0-PSG.edf")
psg   <- prepare_psg(rec) |> preprocess_psg()
hypno <- stage_epochs(psg)$stage

tbp <- compute_temporal_bandpower(psg, hypno)

# Plot delta power across the night
```

```
library(ggplot2)
tbp |>
  dplyr::filter(band == "delta") |>
  ggplot(aes(time_hours, relative_power)) +
  geom_line() +
  labs(x = "Time (hours)", y = "Relative delta power")

## End(Not run)
```

`correct_eog_ica` *Remove EOG artefacts using ICA*

Description

Decomposes the EEG into independent components via FastICA, identifies components whose time courses correlate with the EOG channels above `threshold`, subtracts their contribution from the EEG, and returns a cleaned `mrpheus_psg`. Mirrors MNE's ICA EOG rejection pipeline with default settings matching the companion Python notebook (`n_components = min(6, n_eeg)`, `threshold = 0.35`).

Usage

```
correct_eog_ica(
  psg,
  eog_channels = NULL,
  eeg_channels = NULL,
  n_components = NULL,
  threshold = 0.35,
  fun = "logcosh",
  verbose = TRUE
)
```

Arguments

`psg`	An `mrpheus_psg` object from `prepare_psg()`.
`eog_channels`	Character vector. EOG channel labels used as the artifact reference. If `NULL` (default), all `"EOG"` channels are used.
`eeg_channels`	Character vector. EEG channels to decompose and clean. If `NULL` (default), all non-bad `"EEG"` channels are used.
`n_components`	Integer or `NULL`. Number of ICA components. `NULL` (default) uses `min(6L, n_eeg_channels)`, matching the notebook.
`threshold`	Numeric. Absolute Pearson correlation threshold above which a component is flagged as EOG-related. Default `0.35`.
`fun`	Character. Contrast function passed to `fastICA::fastICA()`. `"logcosh"` (default) or `"exp"`. The R method is used internally for robustness across channel counts; this is slower than the C backend but avoids matrix-conformality errors on small channel sets.
`verbose`	Logical. Print progress messages. Default `TRUE`.

Details

Note: FastICA (this implementation) and MNE's default InfoMax ICA find different decompositions; the identified components and final signal will therefore differ from MNE output, but artifact removal performance is comparable.

Value

A new `mrpheus_psg` with cleaned EEG signals and re-segmented epochs. EOG and other channels are unchanged.

Examples

```
## Not run:
rec   <- read_edf("psg.edf")
psg   <- prepare_psg(rec) |> preprocess_psg()
clean <- correct_eog_ica(psg)

# Stricter threshold
clean <- correct_eog_ica(psg, threshold = 0.5)

## End(Not run)
```

`correct_eog_regression` *Remove EOG artefacts using linear regression*

Description

For each EEG channel, fits a multiple linear regression on all EOG reference channels and subtracts the fitted EOG component, leaving the residual as the cleaned signal. This removes both transient blink artefacts and slow ocular drift without requiring blink event detection.

Usage

```
correct_eog_regression(
  psg,
  eog_channels = NULL,
  eeg_channels = NULL,
  verbose = TRUE
)
```

Arguments

`psg`	An `mrpheus_psg` object from `prepare_psg()`.
`eog_channels`	Character vector. EOG channel labels to use as regressors. If `NULL` (default), all channels of type `"EOG"` in `psg$channel_map` are used.

`eeg_channels`	Character vector. EEG channel labels to clean. If `NULL` (default), all non-bad `"EEG"` channels are used.
`verbose`	Logical. Print progress messages. Default `TRUE`.

Details

This is the pure-R equivalent of MNE's `compute_proj_eog()` / SSP projection pipeline. Both approaches project out the subspace spanned by the EOG channels; regression does so directly on the continuous signal.

Value

A new `mrpheus_psg` with cleaned EEG signals and re-segmented epochs. EOG and other channels are unchanged.

Examples

```
## Not run:
rec   <- read_edf("psg.edf")
psg   <- prepare_psg(rec) |> preprocess_psg()
clean <- correct_eog_regression(psg)

## End(Not run)
```

`detect_apneas` *Detect respiratory events (apneas and hypopneas)*

Description

Detects apneas and hypopneas in respiratory airflow and effort signals, returning event-level metadata and summary indices. Detection follows AASM 2012/2017 criteria: >= 90% signal reduction for >= 10 s (apnea), or >= 30% reduction with >= 3% SpO2 desaturation or arousal for >= 10 s (hypopnea).

Usage

```
detect_apneas(
  psg,
  airflow_channel = NULL,
  spo2_channel = NULL,
  min_duration_s = 10,
  apnea_threshold = 0.9,
  hypopnea_threshold = 0.3,
  desaturation_threshold = 3
)
```

Arguments

`psg`	An `mrpheus_psg` object from `prepare_psg()`.
`airflow_channel`	Character. Airflow channel label (e.g. nasal pressure or thermistor). If `NULL` (default), the first RESP channel is used.
`spo2_channel`	Character or `NULL`. SpO2 channel for hypopnea scoring.
`min_duration_s`	Numeric. Minimum event duration in seconds. Default `10`.
`apnea_threshold`	Numeric. Fractional reduction required for apnea classification. Default `0.90`.
`hypopnea_threshold`	Numeric. Fractional reduction required for hypopnea classification. Default `0.30`.
`desaturation_threshold`	Numeric. SpO2 drop (percentage points) required to confirm hypopnea. Default `3`.

Value

A list with:

events Tibble. One row per event: `epoch`, `start_s`, `end_s`, `duration_s`, `type` (`"apnea"` / `"hypopnea"`), `desaturation`.

summary Tibble. `n_apneas`, `n_hypopneas`, `n_events`.

See Also

`compute_ahi()`, `compute_odi()`

`detect_artifacts` *Detect artefact epochs in a PSG recording*

Description

Flags epochs containing likely artefacts based on amplitude thresholds, excessive high-frequency (muscle) power, and movement contamination. Operates on the EEG channels by default. Returns a logical vector (one value per epoch) and optionally attaches per-epoch artefact metrics for inspection.

Usage

```
detect_artifacts(
  psg,
  channels = NULL,
  amp_threshold_uv = 500,
  hf_band = c(40, 100),
  hf_percentile = 0.99,
  verbose = TRUE
)
```

Arguments

`psg`	An `mrpheus_psg` object from `prepare_psg()`.
`channels`	Character vector. Channel labels to evaluate. If `NULL` (default), all non-bad EEG channels are used.
`amp_threshold_uv`	Numeric. Peak-to-peak amplitude threshold in µV. Epochs exceeding this in any selected channel are flagged. Default `500`.
`hf_band`	Numeric vector of length 2. Frequency band (Hz) considered high-frequency / muscle contamination. Default `c(40, 100)`.
`hf_percentile`	Numeric. Epochs whose HF power exceeds this percentile (across all epochs) are flagged. Default `0.99`.
`verbose`	Logical. Print summary. Default `TRUE`.

Value

A tibble with one row per epoch and columns:

epoch Integer. Epoch index (1-based).

artefact Logical. `TRUE` if the epoch is flagged as artefact.

reason Character. Comma-separated reasons for flagging, or `NA`.

peak_to_peak_uv Numeric. Maximum peak-to-peak amplitude across selected channels.

hf_power_db Numeric. Mean HF band power (dB) across selected channels.

`detect_powerline` *Auto-detect powerline frequency*

Description

Estimates whether dominant powerline interference is at 50 Hz (Europe/UK) or 60 Hz (North America) by comparing PSD power at both frequencies using Welch's method on the first 10 seconds of the signal.

Usage

```
detect_powerline(signal, sr)
```

Arguments

signal	Numeric vector. Should be an EEG or broadband channel with sufficient bandwidth.
sr	Numeric. Sampling rate in Hz. Must be > 100.

Details

If the signal's Nyquist limit is below 60 Hz, returns `50L` immediately.

Value

Integer. `50L` or `60L`.

Examples

```
## Not run:
detect_powerline(eeg_signal, sr = 256)

## End(Not run)
```

---

detect_qrs *Detect QRS Complexes in an ECG Signal (Pan-Tompkins Algorithm)*

Description

Detects R-peaks in a raw ECG signal using the Pan-Tompkins algorithm. Implements bandpass filtering, derivative, squaring, moving-average integration, and adaptive dual-threshold peak detection with T-wave rejection and search-back for missed beats.

Usage

```
detect_qrs(ecg, fs)
```

Arguments

ecg	Numeric vector. Raw ECG signal.
fs	Numeric. Sampling frequency in Hz. If `ecg` comes from `read_philips_physlog()`, use `rec$HDR$sfreq`.

Details

Processing pipeline (Pan & Tompkins, 1985):

1. **Bandpass filter** 5–15 Hz (Butterworth N=3). At fs = 200 Hz, implemented as separate 12 Hz low-pass then 5 Hz high-pass stages.
2. **Derivative filter** — 5-point kernel interpolated to match fs.
3. **Squaring** — non-linear emphasis of dominant peaks.
4. **150 ms moving-average integration.**
5. **Adaptive thresholding** — dual-threshold scheme with T-wave rejection and search-back for missed beats (initialised on 2-second training window).

R-peak indices are returned relative to the bandpass-filtered signal. Account for `delay` if you need indices aligned to the raw ECG.

Value

A list of class `mrpheus_qrs` with components:

`qrs_i` Integer vector of R-peak sample indices (1-based), referenced to the bandpass-filtered signal.

`qrs_amp` Numeric vector of R-wave amplitudes at each detected peak (normalised bandpass signal units).

`delay` Numeric. Sample delay introduced by the moving-average stage (half the 150 ms window length).

`fs` Numeric. Sampling frequency passed to the function.

References

Pan J, Tompkins WJ. A real-time QRS detection algorithm. *IEEE Trans Biomed Eng.* 1985;32(3):230–236. doi:10.1109/TBME.1985.325532

See Also

`read_philips_physlog()`, `compute_hr_signal()`, `compute_hrv_sleep()`

Examples

```
## Not run:
rec  <- read_philips_physlog("sub-01_physlog.log")
qrs  <- detect_qrs(rec$C[, "v1raw"], fs = rec$HDR$sfreq)
qrs

# Downstream HR signal
hr <- compute_hr_signal(qrs)

## End(Not run)
```

`export_hypnogram` *Export a staged hypnogram for use with hypnor*

Description

Prepares the staging tibble produced by `stage_epochs()` for downstream use with the `hypnor` package. Attaches recording metadata as attributes and returns a tibble of class `mrpheus_hypnogram`, which `hypnor::new_hypnogram()` accepts directly once `hypnor` is installed.

Usage

```
export_hypnogram(
  staging,
  epoch_s = 30,
  start_time = NULL,
  participant_id = NULL
)
```

Arguments

`staging`	A tibble from `stage_epochs()` with columns `epoch`, `stage`, and optional probability columns.
`epoch_s`	Numeric. Epoch duration in seconds. Must match the `epoch_s` used in `prepare_psg()`. Default `30`.
`start_time`	POSIXct or `NULL`. Recording start time. Used to compute clock-time axes in `hypnor` plots. If `NULL`, epochs are indexed from 0.
`participant_id`	Character or `NULL`. Optional identifier passed through to `hypnor` and `syncR`.

Value

A tibble of class `mrpheus_hypnogram` with columns `epoch`, `stage`, and any probability columns from the staging model. Metadata (`epoch_s`, `start_time`, `participant_id`, `source`, `resolution`) are attached as attributes and forwarded to `hypnor::new_hypnogram()` when `hypnor` is available.

See Also

`stage_epochs()`

`notch_filter` *Apply a notch (bandstop) filter to a signal*

Description

Designs a 2nd-order Butterworth bandstop filter centred at `freq` Hz with bandwidth `bandwidth_hz` and applies it zero-phase using `gsignal::filtfilt()`. Because the filter has only ~9 coefficients, it is safe to use on long recordings without memory issues.

Usage

```
notch_filter(signal, sr, freq, bandwidth_hz = 2, harmonics = TRUE)
```

Arguments

`signal`	Numeric vector.
`sr`	Numeric. Sampling rate in Hz.
`freq`	Numeric. Centre frequency in Hz (e.g. `50` or `60`).
`bandwidth_hz`	Numeric. Full bandwidth of the notch in Hz. Default `2`.
`harmonics`	Logical. Also notch harmonics (2×, 3×, ...) below Nyquist − 5 Hz. Default `TRUE`.

Details

By default also applies the filter at all harmonics of `freq` below the Nyquist limit, which is the standard approach for powerline noise removal.

Value

Filtered numeric vector, same length as `signal`.

Examples

```
## Not run:
clean <- notch_filter(eeg_signal, sr = 256, freq = 50)
clean <- notch_filter(eeg_signal, sr = 256, freq = 60, harmonics = FALSE)

## End(Not run)
```

`palette_orpheus` *Orpheus mosaic palette*

Description

An 8-colour palette extracted from the Roman mosaic *Orpheus Charming the Animals* (3rd century AD, Palermo Archaeological Museum). The mosaic depicts Orpheus — the mythological figure whose name and story this package honours — surrounded by animals, rendered in warm Mediterranean tesserae.

Usage

```
palette_orpheus
```

Format

A named character vector of 8 hex colour codes:

sand `#CDB992` — warm background tessera
vermillion `#B83E2C` — Orpheus's robe; terracotta
olive `#6A7840` — tree and vegetation
umber `#7C5432` — animal fur and earth
bistre `#3C2212` — shadows and outlines
ochre `#B07C3A` — warm amber accent
slate `#6C8284` — birds; dusty teal-grey
ivory `#EAD6AA` — highlights and light tessera

Details

The palette is intentionally earthy and muted, reflecting the natural pigments of Roman mosaic work: sandy limestone backgrounds, terracotta robes, olive vegetation, warm umber fauna, and the distinctive slate-teal of the birds.

Source

Mosaic: *Orpheus Charming the Animals*, Roman, 3rd century AD. Palermo Archaeological Museum. Image via Wikimedia Commons, https://en.wikipedia.org/wiki/Orpheus.

Examples

```
if (requireNamespace("scales", quietly = TRUE)) {
  scales::show_col(palette_orpheus)
}
```

`prepare_psg` *Prepare a PSG recording for analysis*

Description

Takes an `mrpheus_edf` object and segments it into standard epochs, performs a channel inventory (classifying signals by type), and flags channels that appear flat or likely bad. This is the standard entry point before any downstream analyses (spectral, event detection, staging).

Usage

```
prepare_psg(
  edf,
  epoch_s = 30,
  eeg_pattern = "EEG|C3|C4|F3|F4|O1|O2|Fpz|Pz",
  eog_pattern = "EOG|ROC|LOC",
  emg_pattern = "EMG|chin|Chin",
  ecg_pattern = "ECG|EKG",
  resp_pattern = "Thor|Abdo|Flow|SpO2|airflow",
  flat_threshold = 1e-06
)
```

Arguments

`edf`	An `mrpheus_edf` object from `read_edf()`.
`epoch_s`	Numeric. Epoch length in seconds. Default `30` (standard AASM epoch).
`eeg_pattern`	Character. Regex pattern to identify EEG channels. Default `"EEG|C3|C4|F3|F4|O1|O2|Fpz|Pz"`
`eog_pattern`	Character. Regex pattern to identify EOG channels. Default `"EOG|ROC|LOC"`.
`emg_pattern`	Character. Regex pattern to identify EMG channels. Default `"EMG|chin|Chin"`.
`ecg_pattern`	Character. Regex pattern to identify ECG/EKG channels. Default `"ECG|EKG"`.
`resp_pattern`	Character. Regex pattern to identify respiratory channels. Default `"Thor|Abdo|Flow|SpO2|airf`
`flat_threshold`	Numeric. Variance below this value flags a channel as flat/bad. Default `1e-6`.

Value

A list of class `mrpheus_psg` with components:

edf The original `mrpheus_edf` object.

epochs List. Each element is one epoch (30 s by default), itself a named list of channel vectors.

n_epochs Integer. Total number of complete epochs.

epoch_s Numeric. Epoch duration in seconds.

channel_map Data frame. Channel label, detected type (EEG/EOG/EMG/ECG/RESP/OTHER), sample rate, and `bad` flag.

Examples

```
## Not run:
rec  <- read_edf("data/psg_001.edf")
psg  <- prepare_psg(rec)
psg$channel_map

## End(Not run)
```

`preprocess_psg` *Preprocess a PSG recording*

Description

Applies the standard sleep EEG preprocessing pipeline to an `mrpheus_psg` object: optional channel renaming, DC offset removal, powerline notch filtering (plus harmonics), and channel-type-specific bandpass filtering.

Usage

```
preprocess_psg(
  psg,
  channel_rename = NULL,
  dc = TRUE,
  powerline_freq = NULL,
  notch_harmonics = TRUE,
  notch_bw_hz = 2,
  eeg_bandpass = c(0.3, 35),
  eog_bandpass = c(0.3, 15),
  emg_bandpass = c(10, 99),
  ecg_bandpass = c(0.5, 40),
  verbose = TRUE
)
```

Arguments

`psg`	An `mrpheus_psg` object from `prepare_psg()`.
`channel_rename`	Named character vector mapping old channel labels to new labels, e.g. `c("C3-A2" = "C3", "O2-A1" = "O2")`. Applied before filtering; updates `channel_map` labels accordingly. Default `NULL`.
`dc`	Logical. Remove per-channel DC offset via `remove_dc()`. Default `TRUE`.
`powerline_freq`	Integer or `NULL`. Powerline frequency in Hz (`50L` or `60L`). `NULL` (default) triggers auto-detection via `detect_powerline()` on the first clean EEG channel.

`notch_harmonics`
: Logical. Notch harmonics of `powerline_freq` up to Nyquist - 5 Hz. Default `TRUE`.

`notch_bw_hz`
: Numeric. Full bandwidth of each notch in Hz. Default `2`.

`eeg_bandpass`
: Numeric vector of length 2. Bandpass limits (Hz) applied to all EEG channels. Default `c(0.3, 35)`.

`eog_bandpass`
: Numeric vector of length 2. Bandpass limits (Hz) applied to all EOG channels. Default `c(0.3, 15)`.

`emg_bandpass`
: Numeric vector of length 2. Bandpass limits (Hz) applied to all EMG channels. Default `c(10, 99)`.

`ecg_bandpass`
: Numeric vector of length 2. Bandpass limits (Hz) applied to all ECG channels. Default `c(0.5, 40)`.

`verbose`
: Logical. Print progress messages. Default `TRUE`.

Details

Filtering is applied to the full **continuous** signal stored in `psg$edf$signals` before re-epoching, which avoids filter discontinuities at epoch boundaries. The returned object is a new `mrpheus_psg` with the same epoch length and channel map as the input.

The individual filter steps (`remove_dc()`, `detect_powerline()`, `notch_filter()`, `bandpass_filter()`) are exported separately and can be called on any numeric vector without constructing a PSG object.

Value

A new `mrpheus_psg` object with filtered signals and re-segmented epochs. The `edf$signals` entries are replaced in-place with the filtered continuous signals so that subsequent epoch access is consistent.

Examples

```
## Not run:
rec  <- read_edf("data/psg_001.edf")
psg  <- prepare_psg(rec)

# Auto-detect powerline, apply standard filters
psg_clean <- preprocess_psg(psg)

# Rename linked-ear channels to 10-20 names, fix powerline explicitly
psg_clean <- preprocess_psg(
  psg,
  channel_rename = c("C3-A2" = "C3", "C4-A1" = "C4",
                     "O1-A2" = "O1", "O2-A1" = "O2"),
  powerline_freq = 50L
)

## End(Not run)
```

`read_edf` *Read an EDF or EDF+ recording*

Description

Reads a European Data Format (EDF or EDF+) file and returns a structured list containing signal data, channel metadata, and recording header. Wraps `edfReader::readEdfHeader()` and `edfReader::readEdfSignals()` with consistent output formatting for the mrpheus pipeline.

Usage

```
read_edf(path, channels = NULL, only_header = FALSE)
```

Arguments

`path`	Character. Path to an `.edf` or `.edf+` file.
`channels`	Character vector or `NULL`. Channel labels to import. If `NULL` (default), all channels are imported.
`only_header`	Logical. If `TRUE`, return only the header without reading signal data. Useful for quick channel inspection. Default `FALSE`.

Value

A list of class `mrpheus_edf` with components:

header Data frame. Recording metadata (patient info, start time, number of signals, etc.).

signals Named list of numeric vectors, one per channel.

channels Data frame. Channel-level metadata: label, sample rate, physical min/max, digital min/max, transducer type, prefiltering.

duration_s Numeric. Total recording duration in seconds.

path Character. Resolved path to the source file.

Examples

```
## Not run:
rec <- read_edf("data/psg_001.edf")
rec <- read_edf("data/psg_001.edf", channels = c("EEG Fpz-Cz", "EOG horizontal"))
rec$channels

## End(Not run)
```

`read_philips_physlog` *Read a Philips MRI Physiological Log File*

Description

Reads a Philips PMU physiological log (`.log`) file recorded alongside an MRI acquisition and returns a structured object containing the signal matrix, event markers, and recording metadata. Supports all Philips PMU hardware variants via the `system` preset.

Usage

```
read_philips_physlog(
  path,
  system = c("wBTU", "wired", "custom"),
  sfreq = NULL,
  channels = NULL,
  skipprep = FALSE
)
```

Arguments

`path`	Character. Path to the Philips `.log` file.
`system`	Philips PMU hardware preset. One of:
	`"wBTU"` **(default)** Wireless VCG system, 496 Hz. 10-channel layout: v1raw, v2raw, v1, v2, ppu, resp, gx, gy, gz, mark. Used with the Philips Achieva/Ingenia dStream wireless body telemetry unit.
	`"wired"` Older wired ECG system, 500 Hz. Fewer channels (no accelerometer); layout read from file header.
	`"custom"` Supply `sfreq` explicitly.
`sfreq`	Numeric or `NULL`. Sampling frequency in Hz. Overrides the `system` preset when supplied. Required when `system = "custom"`.
`channels`	Character vector or `NULL`. Channel names to return. `NULL` (default) returns all channels. `"none"` returns no signal data (markers only).
`skipprep`	Logical. If `TRUE`, skip samples recorded before the blank `#` line that marks the end of the preparation phase. Default `FALSE`.

Details

Bit-encoded marker values:

```
0x0001  ECG trigger       0x0002  PPU trigger      0x0004  respiration
0x0008  slice onset       0x0010  scanner start    0x0020  scanner stop
0x0040  external trigger  0x0080  calibration      0x8000  ref ECG trigger
```

Use the scanner-stop marker (more reliable than start) to time-lock data:

```
stopidx  <- tail(data$I$ScannerStop, 1) - enddelay
startidx <- stopidx - round(n_vols * TR * data$HDR$sfreq)
epoch    <- data$C[startidx:stopidx, ]
```

Value

A list of class `mrpheus_physlog` with components:

`C` Integer matrix (samples × channels). Column order matches the file header.

`M` Two-column integer matrix: `marker` (bit-encoded value) and `index` (1-based sample number) for every non-zero marker.

`I` Named list of 1-based sample index vectors for each event type: VcgOnset, PpuOnset, TriggerResp, Measurement, ScannerStart, ScannerStop, TriggerExt, Calibration, RefTriggerVcg.

`HDR` List of header metadata: ID, DATETIME, STATS, DockableTable, COLUMN_NAMES, system, sfreq.

`path` Character. Resolved path to the source file.

See Also

`detect_qrs()`, `compute_hr_signal()`

Examples

```
## Not run:
# Philips Achieva/Ingenia with wireless VCG (default)
rec <- read_philips_physlog("sub-01_ses-01_physlog.log")
rec

# Older wired scanner
rec <- read_philips_physlog("sub-01_physlog.log", system = "wired")

# Custom sampling rate
rec <- read_philips_physlog("sub-01_physlog.log", system = "custom", sfreq = 400)

# ECG channels only
rec <- read_philips_physlog("sub-01_physlog.log", channels = c("v1raw", "v2raw"))

## End(Not run)
```

`remove_dc` *Remove DC offset from a signal*

Description

Subtracts the signal mean (per-channel DC offset removal). Should be applied before filtering to prevent filter transients from a non-zero baseline.

Usage

```
remove_dc(signal)
```

Arguments

signal	Numeric vector.

Value

Numeric vector with mean removed, same length as signal.

Examples

```
x <- c(100.1, 100.5, 99.8, 100.3)
remove_dc(x)
```

---

stage_epochs *Automatic AASM sleep staging*

Description

Stages each 30-second epoch using a pre-trained LightGBM model originally developed for YASA (Vallat & Walker, 2021) and shipped as a cross-language serialised model in inst/models/yasa_staging.txt. Features are computed in R to match the Python feature extraction pipeline exactly; bit-exact parity is validated in the package test suite.

Usage

```
stage_epochs(
  psg,
  eeg_channel = NULL,
  eog_channel = NULL,
  emg_channel = NULL,
  artefacts = NULL,
  model_path = system.file("models", "yasa_staging.txt", package = "mrpheus")
)
```

Arguments

psg	An mrpheus_psg object from prepare_psg().
eeg_channel	Character. Central EEG channel (e.g. "EEG Fpz-Cz"). If NULL (default), the first non-bad EEG channel is used.
eog_channel	Character or NULL. EOG channel. If NULL (default), the first non-bad EOG channel is used (EOG features are omitted if none found).
emg_channel	Character or NULL. Chin EMG channel. If NULL (default), the first non-bad EMG channel is used (EMG features are omitted if none found).

`artefacts` Tibble or `NULL`. Output of `detect_artifacts()`. Artefact epochs are assigned `NA` stage and excluded from the model. If `NULL`, all epochs are staged.

`model_path` Character. Path to the serialised LightGBM model. Defaults to the bundled model at `system.file("models/yasa_staging.txt", package = "mrpheus")`.

Details

Stages returned follow standard AASM nomenclature: `W` (wake), `N1`, `N2`, `N3`, `REM`.

Value

A tibble with one row per epoch:

epoch Integer.

stage Character. AASM stage: `W`, `N1`, `N2`, `N3`, `REM`, or `NA` (artefact).

prob_W, prob_N1, prob_N2, prob_N3, prob_REM Numeric. Posterior class probabilities from the LightGBM model.

References

Vallat, R., & Walker, M. P. (2021). An open-source, high-performance tool for automated sleep staging. *eLife*, 10, e70092. doi:10.7554/eLife.70092

See Also

`export_hypnogram()`

Index

bandpass_filter, 3
bandpass_filter(), *26*

compute_ahi, 4
compute_ahi(), *17*
compute_band_power, 4
compute_band_power(), *13*
compute_hr_signal, 6
compute_hr_signal(), *20*, *29*
compute_hrv_sleep, 7
compute_hrv_sleep(), *6*, *20*
compute_odi, 8
compute_odi(), *17*
compute_slow_oscillations, 8
compute_spectrogram, 10
compute_spindles, 11
compute_temporal_bandpower, 12
correct_eog_ica, 14
correct_eog_regression, 15

detect_apneas, 16
detect_apneas(), *4*
detect_artifacts, 17
detect_artifacts(), *31*
detect_powerline, 18
detect_powerline(), *25*, *26*
detect_qrs, 19
detect_qrs(), *6*, *29*

export_hypnogram, 21
export_hypnogram(), *31*

fastICA::fastICA(), *14*

notch_filter, 22
notch_filter(), *26*

palette_orpheus, 23
prepare_psg, 24
prepare_psg(), *5*, *7–12*, *14*, *15*, *17*, *18*, *21*, *25*, *30*
preprocess_psg, 25

read_edf, 27
read_edf(), *24*
read_philips_physlog, 28
read_philips_physlog(), *19*, *20*
remove_dc, 29
remove_dc(), *25*, *26*

stage_epochs, 30
stage_epochs(), *7*, *21*