

Package: PhysioPreprocess (via r-universe)

October 3, 2026

Title Preprocessing Functions for PhysioExperiment Objects

Version 0.4.1

Description Provides preprocessing functions for physiological signal data stored in 'SummarizedExperiment'-derived objects, including digital filters ('Butterworth' and notch), resampling, re-referencing, artifact detection, independent component analysis ('ICA') based artifact removal, and composable preprocessing pipelines.

Depends R (>= 4.2), PhysioExperiment (>= 2.0.0)

Imports methods, Rcpp, SummarizedExperiment, S4Vectors, signal, stats

Suggests PhysioCore, fastICA, knitr, rmarkdown, testthat (>= 3.2.0)

LinkingTo Rcpp

VignetteBuilder knitr

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Collate 'PhysioPreprocess-package.R' 'RcppExports.R'
'utils-provenance.R' 'filters.R' 'filters-advanced.R'
'filters-causal.R' 'resample.R' 'rereference.R' 'artifact.R'
'asr.R' 'lineNoise.R' 'ica.R' 'detrending.R'
'preprocessing-pipeline.R' 'zzz.R'

RoxygenNote 7.3.3

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

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

Config/pak/sysreqs zlib1g-dev

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

Date/Publication 2026-10-03 13:11:42 UTC

RemoteUrl <https://github.com/x-biosignal/PhysioPreprocess>

RemoteRef HEAD

RemoteSha a00b57e66f266dd6bf36f29a76e32913db9ee2cb

Contents

applyPipeline	3
asrCalibrate	4
asrProcess	5
assaySamplingRates	6
baselineCorrect	6
butterworthFilter	7
classifyICAComponents	9
cleanData	9
cleanLine	11
cleanRawdata	12
createPipeline	13
decimate	14
detectArtifacts	15
detectBadChannels	16
detrendSignal	16
detrendSignals	17
filters-causal	18
filterSignals	19
firFilter	20
getCurrentReference	21
icaDecompose	22
icaRemove	23
interpolate	24
interpolateBadChannels	25
isAverageReferenced	25
lfilter	26
lfilterInit	27
notchFilter	28
print.PhysioPipeline	29
rejectBadEpochs	30
removeBaseline	30
removeICAComponents	31
rereference	32
resample	35
runICA	36
setAssaySamplingRate	37
sosDesign	38
sosfilt	39
sosfiltfilt	40
sosfiltInit	40
StreamFilter	41
zapLine	42

Index

applyPipeline	<i>Apply preprocessing pipeline</i>
---------------	-------------------------------------

Description

Applies a preprocessing pipeline to a `PhysioExperiment` object.

Usage

```
applyPipeline(pe, pipeline, verbose = FALSE)
```

Arguments

<code>pe</code>	A <code>PhysioExperiment</code> object.
<code>pipeline</code>	A pipeline object created with <code>createPipeline()</code> .
<code>verbose</code>	Logical. If <code>TRUE</code> , prints progress messages.

Value

`PhysioExperiment` with all pipeline steps applied.

References

Oppenheim AV, Willsky AS, Nawab SH (1997). "Signals and Systems." 2nd ed. Prentice Hall.

See Also

[createPipeline\(\)](#) for defining the pipeline steps, [butterworthFilter\(\)](#) and [resample\(\)](#) for common preprocessing operations used within pipelines.

Examples

```
pe <- PhysioExperiment(  
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),  
  colData = S4Vectors::DataFrame(label = paste0("Ch", 1:10)),  
  samplingRate = 256  
)  
pipeline <- createPipeline(  
  detrend = list(fn = "detrendSignals", method = "linear")  
)  
pe_processed <- applyPipeline(pe, pipeline)
```

asrCalibrate

*Calibrate Artifact Subspace Reconstruction (ASR)***Description**

Learns the clean-baseline channel covariance and the per-component RMS thresholds that [asrProcess](#) uses to detect and reconstruct artifact subspaces (Mullen et al., 2015). Calibration data should be a relatively clean segment; supply `calib_window` to restrict it.

Usage

```
asrCalibrate(
  x,
  cutoff = 20,
  calib_window = NULL,
  window_len = 0.5,
  assay_name = NULL
)
```

Arguments

<code>x</code>	A <code>PhysioExperiment</code> object with a 2D (time x channels) assay.
<code>cutoff</code>	Rejection cutoff in robust standard deviations; a principal direction whose window RMS exceeds median + cutoff * MAD of the calibration distribution is treated as artifact (default: 20). Inf makes asrProcess an identity.
<code>calib_window</code>	Optional numeric <code>c(start_sec, end_sec)</code> selecting the calibration segment. If NULL (default) the whole signal is used.
<code>window_len</code>	Sliding-window length in seconds (default: 0.5).
<code>assay_name</code>	Input assay (default: <code>defaultAssay(x)</code>).

Value

An object of class "`asr_calibration`": a list with the clean covariance `M`, its square-root mixing matrix, the threshold matrix, per-component thresholds, and settings.

References

Mullen, T. R., et al. (2015). "Real-time neuroimaging and cognitive monitoring using wearable dry EEG." *IEEE Transactions on Biomedical Engineering*, 62(11), 2553-2567. doi:10.1109/TBME.2015.2481482

See Also

[asrProcess](#), [cleanRawdata](#)

asrProcess*Apply Artifact Subspace Reconstruction (ASR)*

Description

Cleans a signal with a calibrated ASR model ([asrCalibrate](#)). For each overlapping sliding window it eigendecomposes the window covariance, flags principal directions whose RMS exceeds the calibrated threshold, and reconstructs those directions from the retained ones via the clean-covariance mixing matrix; windows are recombined by Hann-weighted overlap-add.

Usage

```
asrProcess(  
  x,  
  calibration,  
  window_len = NULL,  
  step = NULL,  
  assay_name = NULL,  
  output_assay = "asr"  
)
```

Arguments

x	A <code>PhysioExperiment</code> object with a 2D (time x channels) assay.
calibration	An "asr_calibration" from asrCalibrate .
window_len	Sliding-window length in seconds (default: the calibration value).
step	Window step in seconds (default: half the window length).
assay_name	Input assay (default: <code>defaultAssay(x)</code>).
output_assay	Name for the cleaned assay (default: "asr").

Value

The `PhysioExperiment` with the cleaned signal in `output_assay`; the removed-variance fraction and rejected-component fraction are stored in `metadata(x)$asr` and logged in provenance.

See Also

[asrCalibrate](#), [cleanRawdata](#)

assaySamplingRates	<i>Per-assay sampling rates (moved to PhysioCore)</i>
--------------------	---

Description

Deprecated: these accessors now live in **PhysioExperiment**. The version here delegates to `PhysioExperiment::assaySamplingRates` for backward compatibility; new code should call the **PhysioExperiment** function directly, or use the canonical multi-rate container `PhysioExperiment::MultiPhysioExperiment()` when streams differ in length.

Usage

```
assaySamplingRates(x)
```

Arguments

`x` A `PhysioExperiment` object.

Value

A named numeric vector of per-assay sampling rates (Hz).

See Also

[assaySamplingRates](#), [MultiPhysioExperiment](#)

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(100 * 2), nrow = 100, ncol = 2)),
  samplingRate = 100
)
assaySamplingRates(pe)
```

baselineCorrect	<i>Baseline correction</i>
-----------------	----------------------------

Description

Subtracts baseline from epochs.

Usage

```
baselineCorrect(
  x,
  baseline = c(-0.2, 0),
  method = c("mean", "median"),
  output_assay = "baseline_corrected"
)
```

Arguments

x	An epoched PhysioExperiment object.
baseline	Numeric vector of length 2 (tmin, tmax) for baseline period.
method	Correction method: "mean" or "median".
output_assay	Name for the output assay.

Value

Modified PhysioExperiment with baseline-corrected data.

butterworthFilter	<i>Butterworth filter</i>
-------------------	---------------------------

Description

Applies a Butterworth filter (lowpass, highpass, bandpass, or bandstop) along the time axis of the specified assay.

Usage

```
butterworthFilter(
  x,
  low = NULL,
  high = NULL,
  order = 4L,
  type = c("pass", "low", "high", "stop"),
  use_sos = TRUE,
  causal = FALSE,
  output_assay = "filtered"
)
```

Arguments

x	A PhysioExperiment object.
low	Lower cutoff frequency in Hz. Required for highpass and bandpass.
high	Upper cutoff frequency in Hz. Required for lowpass and bandpass.
order	Filter order. Default is 4.
type	Filter type: "low", "high", "pass" (bandpass), or "stop" (bandstop).
use_sos	Logical. If TRUE (default), use second-order sections (SOS) form for improved numerical stability, especially for high filter orders. If FALSE, use the traditional transfer function (ba) form.

causal	Logical. If FALSE (default), apply zero-phase forward-backward filtering (signal::filtfilt() / SOS filtfilt()), which is non-causal and symmetrically smears sharp transitions in time. If TRUE, apply a single forward (causal) pass via sosfilt() with a steady-state warm start (sosfiltInit() scaled by the first sample), producing a real-time- equivalent result with a causal group delay and no acausal pre-ringing. Causal filtering always uses SOS form for numerical stability.
output_assay	Name for the output assay. Default is "filtered".

Details

By default, second-order sections (SOS) form is used for filtering, which provides numerical stability for higher filter orders (> 8). The traditional transfer function (ba) form can exhibit coefficient quantization errors at high orders, leading to unstable filters. Set `use_sos = FALSE` to revert to the legacy ba-form behavior.

Value

The input object with a new assay containing filtered data.

See Also

[sosfilt\(\)](#), [StreamFilter\(\)](#) for the underlying causal/stateful filtering primitives.

Examples

```
# Create example EEG data
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000 * 4), nrow = 1000)),
  samplingRate = 250
)

# Bandpass filter (1-40 Hz) - common for EEG
pe <- butterworthFilter(pe, low = 1, high = 40, type = "pass")

# Lowpass filter (30 Hz)
pe <- butterworthFilter(pe, high = 30, type = "low",
  output_assay = "lowpass")

# Highpass filter (0.5 Hz) to remove DC drift
pe <- butterworthFilter(pe, low = 0.5, type = "high",
  output_assay = "highpass")

# High-order filter with SOS (numerically stable)
pe <- butterworthFilter(pe, low = 1, high = 40, type = "pass",
  order = 10)

# Legacy ba-form filtering
pe <- butterworthFilter(pe, low = 1, high = 40, type = "pass",
  use_sos = FALSE, output_assay = "filtered_ba")
```

classifyICAComponents *Classify ICA components as brain or artifact*

Description

Automatically identifies artifact ICA components using statistical criteria.

Usage

```
classifyICAComponents(
  x,
  method = c("autocorrelation", "kurtosis", "frequency"),
  eog_channels = NULL,
  threshold = NULL
)
```

Arguments

x	A <code>PhysioExperiment</code> object with ICA decomposition (from icaDecompose).
method	Classification method: "autocorrelation", "kurtosis", or "frequency".
eog_channels	Integer vector of EOG channel indices for correlation-based detection. If provided, also computes EOG correlation scores.
threshold	Classification threshold. If NULL, uses method-specific defaults: autocorrelation=0.9, kurtosis=3.0, frequency=0.5.

Value

A list with:

- `artifact_indices`: Integer vector of artifact component indices
- `scores`: Named numeric vector of scores per component
- `threshold`: The threshold used
- `method`: The method used

cleanData	<i>Clean data using an artifact removal pipeline</i>
-----------	--

Description

Runs a configurable sequence of artifact removal steps.

Usage

```
cleanData(  
  x,  
  steps = c("bad_channels", "ica"),  
  bad_channel_method = "zscore",  
  bad_channel_threshold = NULL,  
  interpolation_method = "average",  
  ica_method = "fastica",  
  ica_classify_method = "kurtosis",  
  output_assay = "cleaned"  
)
```

Arguments

x	A <code>PhysioExperiment</code> object.
steps	Character vector of steps to run. Options: "bad_channels", "ica".
bad_channel_method	Method for detectBadChannels .
bad_channel_threshold	Threshold for bad channel detection.
interpolation_method	Method for interpolateBadChannels .
ica_method	Method for icaDecompose .
ica_classify_method	Method for classifyICAComponents .
output_assay	Name for the output assay.

Value

Modified `PhysioExperiment` with cleaned data.

Examples

```
pe <- PhysioExperiment(  
  assays = list(raw = matrix(rnorm(300 * 3), nrow = 300, ncol = 3)),  
  samplingRate = 100  
)  
# Bad-channel detection/interpolation only (ICA needs the fastICA package)  
pe <- cleanData(pe, steps = "bad_channels")
```

cleanLine	<i>Remove line noise with CleanLine</i>
-----------	---

Description

Removes power-line noise (and optional harmonics) by sliding-window sinusoidal regression with a Thomson F-test (the CleanLine / Chronux approach; Mitra & Bokil, 2008). In each Hann-overlapped window each channel is regressed onto the line sine/cosine pair; when the fit is significant ($p < p_thresh$) the fitted sinusoid is subtracted. This attenuates the line deeply without a wide spectral notch and works on any number of channels.

Usage

```
cleanLine(
  x,
  line_freq = 50,
  bandwidth = 2,
  harmonics = 1,
  p_thresh = 0.01,
  window_sec = 4,
  assay_name = NULL,
  output_assay = "cleanline"
)
```

Arguments

x	A PhysioExperiment object with a 2D (time x channels) assay.
line_freq	Line frequency in Hz (default: 50).
bandwidth	Reserved spectral bandwidth in Hz around the line (default: 2).
harmonics	Number of harmonics to remove, including the fundamental (default: 1, i.e. line_freq only; 3 removes 50/100/150 Hz).
p_thresh	F-test significance threshold for subtracting the line (default: 0.01).
window_sec	Sliding-window length in seconds (default: 4).
assay_name	Input assay (default: defaultAssay(x)).
output_assay	Name for the cleaned assay (default: "cleanline").

Value

The PhysioExperiment with the cleaned signal in output_assay; the removed line-power fraction is stored in metadata(x)\$line_noise and logged in provenance.

References

Mitra, P., & Bokil, H. (2008). *Observed Brain Dynamics*. Oxford University Press.

See Also

[zapLine](#), [notchFilter](#) (the fast fixed notch)

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400 * 3), nrow = 400, ncol = 3)),
  samplingRate = 200
)
pe <- cleanLine(pe, line_freq = 50, window_sec = 1)
```

cleanRawdata

Automated raw-data cleaning (EEGLAB clean_rawdata-style)

Description

Orchestrates a standard cleaning pipeline mirroring EEGLAB's clean_rawdata: flat-channel removal, high-pass drift removal, correlation-based bad-channel interpolation, and Artifact Subspace Reconstruction. The removed-variance fraction and any repaired channels are recorded in provenance.

Usage

```
cleanRawdata(
  x,
  flatline = TRUE,
  highpass = c(0.25, 0.75),
  channel_crit = 0.8,
  asr_cutoff = 20,
  window_len = 0.5,
  calib_window = NULL,
  assay_name = NULL,
  output_assay = "clean"
)
```

Arguments

x	A PhysioExperiment object with a 2D (time x channels) assay.
flatline	Logical; detect and interpolate (near-)flat channels (default: TRUE).
highpass	Numeric c(low, high) transition band in Hz for the drift-removal high-pass; the cutoff is the band midpoint (default c(0.25, 0.75)). NULL skips high-pass filtering.
channel_crit	Minimum mean absolute correlation with the other channels for a channel to be kept; channels below this are interpolated (default: 0.8). NULL skips the correlation criterion.
asr_cutoff	ASR rejection cutoff in robust SDs (default: 20). Inf or NULL skips ASR.

window_len	ASR sliding-window length in seconds (default: 0.5).
calib_window	Optional c(start_sec, end_sec) calibration segment for ASR.
assay_name	Input assay (default: defaultAssay(x)).
output_assay	Name for the cleaned assay (default: "clean").

Value

The PhysioExperiment with the cleaned signal in output_assay; metadata(x)\$clean_rawdata records the interpolated channels and the removed-variance fraction, also logged in provenance.

References

Mullen, T. R., et al. (2015). "Real-time neuroimaging and cognitive monitoring using wearable dry EEG." *IEEE Transactions on Biomedical Engineering*, 62(11), 2553-2567. doi:10.1109/TBME.2015.2481482

See Also

[asrCalibrate](#), [asrProcess](#)

createPipeline	<i>Preprocessing Pipeline for PhysioExperiment</i>
----------------	--

Description

Functions for creating and applying preprocessing pipelines. Create preprocessing pipeline

Usage

```
createPipeline(...)
```

Arguments

... Preprocessing steps as named function calls.

Details

Creates a preprocessing pipeline specification that can be applied to data.

Value

A preprocessing pipeline object (list).

References

Oppenheim AV, Willsky AS, Nawab SH (1997). "Signals and Systems." 2nd ed. Prentice Hall.

See Also

[applyPipeline\(\)](#) for executing a pipeline on data, [filterSignals\(\)](#) and [detrendSignals\(\)](#) for individual preprocessing steps that can be included in a pipeline.

Examples

```
pipeline <- createPipeline(
  filter = list(fn = "filterSignals", lowcut = 1, highcut = 40),
  detrend = list(fn = "detrendSignals", method = "linear")
)
```

decimate

*Decimate signal***Description**

Downsamples by an integer factor with anti-aliasing filter. An 80%-Nyquist lowpass Butterworth filter is applied before decimation to prevent aliasing.

Usage

```
decimate(x, factor, filter_order = 8L, output_assay = "decimated")
```

Arguments

x	A <code>PhysioExperiment</code> object.
factor	Integer decimation factor.
filter_order	Order of the anti-aliasing lowpass filter.
output_assay	Name for the output assay.

Value

A new `PhysioExperiment` object with sampling rate equal to the original rate divided by factor. The time dimension is reduced accordingly. Column data and metadata are carried over.

References

Crochiere, R.E. & Rabiner, L.R. (1983). "Multirate Digital Signal Processing." Prentice Hall.

See Also

[resample\(\)](#) for arbitrary-rate resampling, [interpolate\(\)](#) for integer-factor upsampling, [butterworthFilter\(\)](#) for the anti-aliasing filter used internally.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400 * 2), nrow = 400, ncol = 2)),
  samplingRate = 100
)
pe_dec <- decimate(pe, factor = 2)
samplingRate(pe_dec)
```

detectArtifacts	<i>Detect artifacts in continuous data</i>
-----------------	--

Description

Scans continuous data for artifacts using amplitude or gradient criteria.

Usage

```
detectArtifacts(
  x,
  method = c("amplitude", "gradient", "joint"),
  threshold = NULL,
  window_sec = 1
)
```

Arguments

x	A PhysioExperiment object.
method	Detection method: "amplitude", "gradient", or "joint".
threshold	Detection threshold. If NULL, computed as median + 5 * MAD.
window_sec	Detection window in seconds (default: 1.0).

Value

A data.frame with columns: onset (seconds), offset (seconds), channel (index), type (detection method).

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(500 * 2), nrow = 500, ncol = 2)),
  samplingRate = 100
)
arts <- detectArtifacts(pe, method = "amplitude")
head(arts)
```

detectBadChannels	<i>Detect bad channels</i>
-------------------	----------------------------

Description

Identifies channels with abnormal characteristics.

Usage

```
detectBadChannels(
  x,
  method = c("zscore", "correlation", "flatline"),
  threshold = NULL
)
```

Arguments

x	A PhysioExperiment object.
method	Detection method: "zscore", "correlation", or "flatline".
threshold	Threshold for detection (depends on method).

Value

Integer vector of bad channel indices.

Examples

```
m <- matrix(rnorm(300 * 3), nrow = 300, ncol = 3)
m[, 2] <- m[, 2] * 10 # channel 2 is an amplitude outlier
pe <- PhysioExperiment(assays = list(raw = m), samplingRate = 100)
detectBadChannels(pe, method = "zscore")
```

detrendSignal	<i>Detrend signal</i>
---------------	-----------------------

Description

Removes linear or polynomial trends from the signal.

Usage

```
detrendSignal(x, type = c("linear", "constant"), output_assay = "detrended")
```

Arguments

`x` A `PhysioExperiment` object.

`type` Type of detrending: "linear" or "constant" (mean removal).

`output_assay` Name for the output assay. Default is "detrended".

Value

A `PhysioExperiment` object with a new assay named `output_assay` containing detrended data. For "constant" type, the channel mean is subtracted. For "linear" type, a least-squares linear fit is removed.

References

Oppenheim, A.V. & Willsky, A.S. (1997). "Signals and Systems." 2nd ed. Prentice Hall.

See Also

[detrendSignals\(\)](#) for the alternative detrending implementation with polynomial support, [butterworthFilter\(\)](#) for highpass filtering as an alternative to detrending.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(200 * 2) + seq_len(200) / 50,
                             nrow = 200, ncol = 2)),
  samplingRate = 100
)
pe <- detrendSignal(pe, type = "linear")
```

detrendSignals

Signal Detrending Functions for PhysioExperiment

Description

Functions for removing trends from physiological signals. Detrend signals

Usage

```
detrendSignals(
  pe,
  method = c("linear", "mean", "polynomial"),
  order = 2,
  assay_name = NULL,
  output_assay = "detrended"
)
```

Arguments

pe	A <code>PhysioExperiment</code> object.
method	Detrending method: "linear", "mean", or "polynomial".
order	Polynomial order for method="polynomial".
assay_name	Name of the assay to detrend.
output_assay	Name for the detrended output assay.

Details

Removes linear or polynomial trends from signals.

Value

`PhysioExperiment` with detrended data in `output_assay`.

References

Oppenheim AV, Willsky AS, Nawab SH (1997). "Signals and Systems." 2nd ed. Prentice Hall.

See Also

[removeBaseline\(\)](#) for baseline subtraction over a time window, [detrendSignal\(\)](#) for the alternative linear/constant detrending implementation, [butterworthFilter\(\)](#) for highpass filtering as an alternative to detrending.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000) + 1:100/10, nrow = 100, ncol = 10)),
  colData = S4Vectors::DataFrame(label = paste0("Ch", 1:10)),
  samplingRate = 256
)
pe_detrended <- detrendSignals(pe, method = "linear")
```

Description

This file provides one-pass, causal, state-carrying IIR filtering that mirrors `scipy.signal` (`lfilter`, `lfilter_zi`, `sosfilt`, `sosfilt_zi`, `sosfiltfilt`). Unlike the zero-phase [butterworthFilter](#) default (which uses forward-backward `filtfilt` and is inherently non-causal), these functions process a signal strictly forward in time and can carry filter state across consecutive chunks. This is the numerical core that a real-time streaming backend (e.g. a future `PhysioStream`) reuses so that offline and online filtering produce identical results.

Details

All recursions use the Direct-Form-II-Transposed (DF2T) structure, which is the same structure `scipy.signal` and `signal::filter` use, so the outputs agree to machine precision.

References

Oppenheim, A.V. & Schafer, R.W. (2010). *Discrete-Time Signal Processing*, 3rd ed. Prentice Hall.

Virtanen, P. et al. (2020). SciPy 1.0. *Nature Methods* 17, 261-272. (reference implementation: `scipy.signal.sosfilt / lfilter / lfilter_zi / sosfilt_zi`).

filterSignals	<i>Moving average filter</i>
---------------	------------------------------

Description

Applies a moving average filter along the first dimension (time axis) of the default assay.

Usage

```
filterSignals(x, window = 5L, na.rm = FALSE, output_assay = "filtered")
```

Arguments

<code>x</code>	A <code>PhysioExperiment</code> object.
<code>window</code>	Integer window length for the moving average.
<code>na.rm</code>	Logical. If TRUE, NA values are ignored in the filter computation.
<code>output_assay</code>	Name for the output assay. Default is "filtered".

Value

A `PhysioExperiment` object with a new assay named `output_assay` containing the moving-average-filtered data. Dimensions match the input assay. Edge values where the full window cannot be applied are set to NA.

References

Oppenheim, A.V. & Willsky, A.S. (1997). "Signals and Systems." 2nd ed. Prentice Hall.

See Also

[butterworthFilter\(\)](#) for IIR filtering, [firFilter\(\)](#) for FIR filtering, [notchFilter\(\)](#) for power line noise removal, [detrendSignal\(\)](#) for trend removal.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(300 * 3), nrow = 300, ncol = 3)),
  samplingRate = 100
)
pe <- filterSignals(pe, window = 5)
```

firFilter

FIR filter

Description

Applies a Finite Impulse Response (FIR) filter along the time axis. Uses zero-phase forward-backward filtering via `signal::filtfilt()` to avoid phase distortion.

Usage

```
firFilter(
  x,
  low = NULL,
  high = NULL,
  order = 100L,
  type = c("pass", "low", "high", "stop"),
  window = "hamming",
  causal = FALSE,
  output_assay = "filtered"
)
```

Arguments

x	A PhysioExperiment object.
low	Lower cutoff frequency in Hz.
high	Upper cutoff frequency in Hz.
order	Filter order (number of taps - 1). Default is 100.
type	Filter type: "low", "high", "pass" (bandpass), or "stop" (bandstop).
window	Window function for FIR design. Default is "hamming".
causal	Logical. If FALSE (default), apply zero-phase forward-backward filtering (<code>signal::filtfilt()</code>), which is non-causal. If TRUE, apply a single forward (causal) pass via <code>lfilter()</code> with a steady-state warm start, producing a real-time-equivalent result (linear-phase group delay of order / 2 samples, no acausal pre-ringing).
output_assay	Name for the output assay. Default is "filtered".

Value

A PhysioExperiment object with a new assay named output_assay containing the FIR-filtered data. Dimensions match the input assay.

References

Oppenheim, A.V. & Willsky, A.S. (1997). "Signals and Systems." 2nd ed. Prentice Hall.

See Also

[butterworthFilter\(\)](#) for IIR filtering, [notchFilter\(\)](#) for power line noise removal, [filterSignals\(\)](#) for moving average filtering, [lfilter\(\)](#) for the underlying causal filtering primitive.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(600 * 2), nrow = 600, ncol = 2)),
  samplingRate = 100
)
# Linear-phase FIR bandpass (1-20 Hz); modest order keeps the example quick
pe <- firFilter(pe, low = 1, high = 20, order = 20, type = "pass")
```

getCurrentReference	<i>Get current reference</i>
---------------------	------------------------------

Description

Returns the current reference electrode information.

Usage

```
getCurrentReference(x)
```

Arguments

x A PhysioExperiment object.

Value

A character string describing the current reference (e.g., "average", "Cz", "M1+M2"), or NULL if no reference has been set.

References

Nunez, P.L. & Srinivasan, R. (2006). "Electric Fields of the Brain." 2nd ed. Oxford University Press.

See Also

[rereference\(\)](#) for changing the reference, [isAverageReferenced\(\)](#) for checking average reference status.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(100), nrow = 10, ncol = 10)),
  samplingRate = 100
)
pe <- setReference(pe, "Cz")
getCurrentReference(pe) # "Cz"
```

icaDecompose

ICA and Artifact Removal for PhysioExperiment

Description

Functions for Independent Component Analysis (ICA) and artifact removal from physiological signals.

Usage

```
icaDecompose(
  x,
  n_components = NULL,
  method = c("fastica", "jade"),
  max_iter = 200L,
  tol = 1e-04
)
```

Arguments

<code>x</code>	A <code>PhysioExperiment</code> object.
<code>n_components</code>	Number of components to extract. If <code>NULL</code> , uses number of channels.
<code>method</code>	ICA method: "fastica" (default) or "jade".
<code>max_iter</code>	Maximum iterations for convergence.
<code>tol</code>	Tolerance for convergence.

Value

A list with four elements:

components The independent components as a matrix or 3D array (time by component by samples for 3D data).

mixing The mixing matrix (channels x components).

unmixing The unmixing matrix (components x channels).

object The input `PhysioExperiment` with the ICA decomposition (components, mixing/unmixing matrices, column means) stored in `metadata(x)$ica`. The components are held in metadata rather than as an assay because their column count is `n_components`, which need not equal the object's channel count.

References

Hyvarinen, A. & Oja, E. (2000). "Independent component analysis: algorithms and applications." *Neural Networks*, 13(4-5), 411-430. doi:10.1016/S0893-6080(00)00026-5 Perform ICA decomposition

Decomposes the signal into independent components using FastICA algorithm.

Hyvarinen, A. & Oja, E. (2000). "Independent component analysis: algorithms and applications." *Neural Networks*, 13(4-5), 411-430. doi:10.1016/S0893-6080(00)00026-5

See Also

[icaRemove\(\)](#) for removing specific components after decomposition, [runICA\(\)](#) for an alternative ICA implementation using the fastICA package, [detectBadChannels\(\)](#) for channel-level artifact detection.

icaRemove	<i>Remove ICA components</i>
-----------	------------------------------

Description

Removes specified ICA components and reconstructs the signal.

Usage

```
icaRemove(x, components, output_assay = "ica_cleaned")
```

Arguments

x	A <i>PhysioExperiment</i> object with ICA decomposition.
components	Integer vector of component indices to remove.
output_assay	Name for the output assay.

Value

Modified *PhysioExperiment* with cleaned signal.

interpolate	<i>Interpolate signal</i>
-------------	---------------------------

Description

Upsamples by an integer factor with interpolation. This is a convenience wrapper around [resample\(\)](#) with `target_rate = sr * factor`.

Usage

```
interpolate(
  x,
  factor,
  method = c("linear", "spline"),
  output_assay = "interpolated"
)
```

Arguments

<code>x</code>	A <code>PhysioExperiment</code> object.
<code>factor</code>	Integer interpolation factor.
<code>method</code>	Interpolation method: "linear" or "spline".
<code>output_assay</code>	Name for the output assay.

Value

A new `PhysioExperiment` object with sampling rate equal to the original rate multiplied by `factor`. The time dimension is increased accordingly.

References

Crochiere, R.E. & Rabiner, L.R. (1983). "Multirate Digital Signal Processing." Prentice Hall.

See Also

[resample\(\)](#) for arbitrary-rate resampling, [decimate\(\)](#) for integer-factor downsampling.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(200 * 2), nrow = 200, ncol = 2)),
  samplingRate = 100
)
pe_up <- interpolate(pe, factor = 2)
samplingRate(pe_up)
```

```
interpolateBadChannels
```

Interpolate bad channels

Description

Replaces bad channels with interpolated values from neighboring channels.

Usage

```
interpolateBadChannels(
  x,
  bad_channels,
  method = c("average", "spline"),
  output_assay = "interpolated"
)
```

Arguments

<code>x</code>	A <code>PhysioExperiment</code> object.
<code>bad_channels</code>	Integer vector of channel indices to interpolate.
<code>method</code>	Interpolation method: "average" or "spline".
<code>output_assay</code>	Name for the output assay.

Value

Modified `PhysioExperiment` with interpolated channels.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(300 * 3), nrow = 300, ncol = 3)),
  samplingRate = 100
)
pe <- interpolateBadChannels(pe, bad_channels = 2, method = "average")
```

```
isAverageReferenced
```

Check if data is average referenced

Description

Check if data is average referenced

Usage

```
isAverageReferenced(x)
```

Arguments

`x` A `PhysioExperiment` object.

Value

A logical scalar: TRUE if the reference metadata is set to "average", FALSE otherwise.

References

Nunez, P.L. & Srinivasan, R. (2006). "Electric Fields of the Brain." 2nd ed. Oxford University Press.

See Also

[rereference\(\)](#) for changing the reference, [getCurrentReference\(\)](#) for querying the current reference.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(100), nrow = 10, ncol = 10)),
  samplingRate = 100
)
pe <- rereference(pe, ref_type = "average")
isAverageReferenced(pe) # TRUE
```

lfilter

One-pass causal IIR filtering (Direct Form II Transposed)

Description

Applies a rational transfer function defined by numerator `b` and denominator `a` to `x`, forward in time, mirroring `scipy.signal.lfilter`. Optionally carries the delay-line state so a signal can be filtered in consecutive chunks with a result identical to filtering it whole.

Usage

```
lfilter(b, a, x, zi = NULL)
```

Arguments

`b` Numerator (feed-forward) coefficients.

`a` Denominator (feedback) coefficients (`a[1]` normalised internally if not 1).

`x` Numeric input vector.

`zi` Optional initial delay-line state, length `max(length(a), length(b)) - 1`. When supplied, the return value is a list carrying the final state; when `NULL` (default) the state starts at zero and a plain filtered vector is returned. Use [lfilterInit](#) for a transient-free warm start.

Value

If `zi` is `NULL`, the filtered numeric vector. Otherwise a list with `y` (filtered vector) and `zi` (final state).

See Also

[lfilterInit](#), [sosfilt](#)

Examples

```
ba <- signal::butter(4, 0.2, "low")
# single call
y <- lfilter(ba$b, ba$a, rnorm(500))
# streamed in two chunks with carried state
x <- rnorm(500)
r1 <- lfilter(ba$b, ba$a, x[1:250], zi = numeric(4))
r2 <- lfilter(ba$b, ba$a, x[251:500], zi = r1$zi)
identical(length(c(r1$y, r2$y)), length(x))
```

lfilterInit

Steady-state initial conditions for a one-pass IIR filter

Description

Computes the filter delay-line state that, when scaled by the first input sample and passed to [lfilter](#), makes the filter's step response start already in steady state (no start-up transient). This is the direct analogue of `scipy.signal.lfilter_zi`.

Usage

```
lfilterInit(b, a)
```

Arguments

<code>b</code>	Numerator (feed-forward) coefficients.
<code>a</code>	Denominator (feedback) coefficients; <code>a[1]</code> need not be 1 (coefficients are normalised internally).

Value

A numeric vector of length `max(length(a), length(b)) - 1`, the initial delay-line state (unscaled; multiply by the first sample level).

See Also

[lfilter](#), [sosfiltInit](#)

Examples

```
ba <- signal::butter(2, 0.2, "low")
zi <- lfilterInit(ba$b, ba$a)
```

notchFilter

*Notch filter (power line noise removal)***Description**

Applies a notch filter to remove power line noise (50 Hz or 60 Hz) and optionally its harmonics. Implemented as a Butterworth bandstop filter with zero-phase filtering.

Usage

```
notchFilter(
  x,
  freq = 50,
  bandwidth = 2,
  harmonics = 1L,
  output_assay = "filtered"
)
```

Arguments

x	A <code>PhysioExperiment</code> object.
freq	Center frequency to remove in Hz. Default is 50 (European power line).
bandwidth	Bandwidth of the notch in Hz. Default is 2.
harmonics	Number of harmonics to remove. Default is 1 (only fundamental).
output_assay	Name for the output assay. Default is "filtered".

Value

A `PhysioExperiment` object with a new assay named `output_assay` containing the notch-filtered data. Harmonics above the Nyquist frequency are skipped with a warning.

References

Oppenheim, A.V. & Willsky, A.S. (1997). "Signals and Systems." 2nd ed. Prentice Hall.

See Also

[butterworthFilter\(\)](#) for general Butterworth filtering, [firFilter\(\)](#) for FIR filtering, [filterSignals\(\)](#) for moving average filtering.

Examples

```
pe <- PhysioExperiment(  
  assays = list(raw = matrix(rnorm(1000 * 4), nrow = 1000)),  
  samplingRate = 250  
)  
  
# Remove 50 Hz power line noise (Europe/Asia)  
pe <- notchFilter(pe, freq = 50)  
  
# Remove 60 Hz and harmonics (Americas)  
pe <- notchFilter(pe, freq = 60, harmonics = 2)
```

```
print.PhysioPipeline  Print pipeline summary
```

Description

Print pipeline summary

Usage

```
## S3 method for class 'PhysioPipeline'  
print(x, ...)
```

Arguments

x	A PhysioPipeline object.
...	Additional arguments (ignored).

Value

Invisible x.

See Also

[createPipeline\(\)](#) for creating pipelines, [applyPipeline\(\)](#) for executing pipelines.

rejectBadEpochs	<i>Reject bad epochs</i>
-----------------	--------------------------

Description

Identifies and optionally removes epochs with artifacts.

Usage

```
rejectBadEpochs(  
  x,  
  threshold = 100,  
  method = c("amplitude", "gradient", "variance"),  
  remove = TRUE  
)
```

Arguments

x	An epoched PhysioExperiment object.
threshold	Amplitude threshold for rejection.
method	Detection method: "amplitude", "gradient", or "variance".
remove	If TRUE, removes bad epochs. If FALSE, returns indices only.

Value

If remove=TRUE, modified object. If remove=FALSE, indices of bad epochs.

removeBaseline	<i>Remove baseline</i>
----------------	------------------------

Description

Subtracts baseline from a specified time window.

Usage

```
removeBaseline(  
  pe,  
  baseline_start,  
  baseline_end,  
  assay_name = NULL,  
  output_assay = "baseline_corrected"  
)
```

Arguments

pe A `PhysioExperiment` object.
 baseline_start Start time of baseline window in seconds.
 baseline_end End time of baseline window in seconds.
 assay_name Name of the assay to baseline correct.
 output_assay Name for the baseline-corrected output assay.

Value

PhysioExperiment with baseline-corrected data.

References

Oppenheim AV, Willsky AS, Nawab SH (1997). "Signals and Systems." 2nd ed. Prentice Hall.

See Also

[detrendSignals\(\)](#) for trend removal, [baselineCorrect\(\)](#) for epoch-based baseline correction,
[filterSignals\(\)](#) for moving average smoothing.

Examples

```

pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000) + 5, nrow = 100, ncol = 10)),
  colData = S4Vectors::DataFrame(label = paste0("Ch", 1:10)),
  samplingRate = 100
)
pe_corrected <- removeBaseline(pe, baseline_start = 0, baseline_end = 0.2)

```

removeICAComponents	<i>Remove ICA components</i>
---------------------	------------------------------

Description

Reconstructs signals after removing specified ICA components (e.g., artifacts).

Usage

```

removeICAComponents(
  pe,
  ica_result,
  remove_components,
  assay_name = NULL,
  output_assay = "ica_cleaned"
)

```

Arguments

<code>pe</code>	A <code>PhysioExperiment</code> object.
<code>ica_result</code>	Result from <code>runICA()</code> .
<code>remove_components</code>	Integer vector of component indices to remove.
<code>assay_name</code>	Name of the assay to reconstruct.
<code>output_assay</code>	Name for the cleaned output assay.

Value

`PhysioExperiment` with cleaned data in `output_assay`.

References

Hyvarinen A, Oja E (2000). "Independent component analysis: algorithms and applications." *Neural Networks*, 13(4-5), 411-430.

See Also

[runICA\(\)](#) for performing the ICA decomposition, [icaRemove\(\)](#) for the alternative component removal implementation, [detectBadChannels\(\)](#) for channel-level artifact detection.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  colData = S4Vectors::DataFrame(label = paste0("Ch", 1:10)),
  samplingRate = 256
)

if (requireNamespace("fastICA", quietly = TRUE)) {
  ica_result <- runICA(pe, n_components = 5)
  pe_clean <- removeICAComponents(pe, ica_result, remove_components = c(1, 3))
}
```

rereference

Re-referencing Operations for EEG Data

Description

Functions for changing the reference electrode in EEG recordings. Re-referencing is a common preprocessing step that affects the spatial distribution of the signal.

Usage

```

rereference(
  x,
  ref_type = c("average", "robust", "median", "channel", "channels", "REST"),
  ref_channels = NULL,
  exclude = NULL,
  input_assay = NULL,
  output_assay = "rereferenced",
  keep_ref = TRUE,
  robust_noise_sd = 4,
  robust_max_iter = 5
)

```

Arguments

<code>x</code>	A <code>PhysioExperiment</code> object.
<code>ref_type</code>	Type of re-referencing: "average" (common average reference), "robust" (PREP robust average reference: iteratively detect and exclude high-variance bad channels from the average; Bigdely-Shamlo et al., 2015), "median" (channel-wise median reference, robust to outlier channels), "channel" (single channel), "channels" (average of specified channels), or "REST" (Reference Electrode Standardization Technique).
<code>ref_channels</code>	For "channel" or "channels" type, the channel name(s) or index/indices to use as reference.
<code>exclude</code>	Channels to exclude from average reference calculation (e.g., non-EEG channels like EOG, EMG).
<code>input_assay</code>	Input assay name. If <code>NULL</code> , uses default assay.
<code>output_assay</code>	Output assay name. Default is "rereferenced".
<code>keep_ref</code>	Logical. If <code>TRUE</code> , keeps the original reference channel(s) in the output (zeroed). If <code>FALSE</code> , removes them.
<code>robust_noise_sd</code>	For "robust", the threshold (robust SDs above the median channel variance) for flagging a bad channel (default: 4).
<code>robust_max_iter</code>	For "robust", the maximum number of PREP iterations (default: 5).

Details

Re-referencing transforms the data by subtracting a reference signal from each channel. The choice of reference affects the spatial distribution and interpretation of the signal.

Average reference ("average"): Subtracts the mean of all channels at each time point. This is commonly used for high-density EEG and provides a reference-independent measure, but requires good spatial sampling.

Single channel reference ("channel"): Subtracts the signal from a specified electrode. Common choices include Cz, linked mastoids (A1+A2)/2, or nose reference.

Multi-channel reference ("channels"): Subtracts the average of multiple specified channels. Useful for linked mastoids or other custom references.

resample

*Resampling operations for PhysioExperiment***Description**

Functions for resampling signal data to different sampling rates. Resample signal data

Usage

```
resample(
  x,
  target_rate,
  method = c("linear", "spline", "fft"),
  assay_name = NULL,
  output_assay = "resampled"
)
```

Arguments

<code>x</code>	A <code>PhysioExperiment</code> object.
<code>target_rate</code>	Target sampling rate in Hz.
<code>method</code>	Resampling method: "linear" (default), "spline", or "fft".
<code>assay_name</code>	Optional assay name. If NULL, uses the default assay.
<code>output_assay</code>	Name for the output assay. Default is "resampled".

Details

Resamples the signal data to a target sampling rate using interpolation. Supports linear interpolation, spline interpolation, and FFT-based resampling (zero-padding or truncation in the frequency domain).

Value

A new `PhysioExperiment` object with sampling rate set to `target_rate`. The time dimension is adjusted to match the new rate while preserving the signal duration. Column data and metadata are carried over from the input.

References

Crochiere, R.E. & Rabiner, L.R. (1983). "Multirate Digital Signal Processing." Prentice Hall.

See Also

[decimate\(\)](#) for integer-factor downsampling with anti-aliasing, [interpolate\(\)](#) for integer-factor upsampling, [setAssaySamplingRate\(\)](#) for per-assay rate tracking.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(500 * 2), nrow = 500, ncol = 2)),
  samplingRate = 100
)
pe_50 <- resample(pe, target_rate = 50)
samplingRate(pe_50)
```

runICA

*ICA-Based Artifact Removal for PhysioExperiment***Description**

Functions for Independent Component Analysis (ICA) decomposition and artifact removal using the fastICA algorithm. Perform ICA decomposition

Usage

```
runICA(
  pe,
  n_components = NULL,
  assay_name = NULL,
  method = "fastica",
  max_iter = 200,
  ...
)
```

Arguments

pe	A PhysioExperiment object.
n_components	Number of components to extract. If NULL, uses all channels.
assay_name	Name of the assay to decompose.
method	ICA algorithm: "fastica" (default).
max_iter	Maximum number of iterations.
...	Additional arguments passed to fastICA::fastICA.

Details

Decomposes signals into independent components using the fastICA algorithm.

Value

A list with components:

S	Source matrix (time x components)
A	Mixing matrix (channels x components)
W	Unmixing matrix (components x channels)

References

Hyvarinen A, Oja E (2000). "Independent component analysis: algorithms and applications." *Neural Networks*, 13(4-5), 411-430.

See Also

[removeICAComponents\(\)](#) for reconstructing signals after removing artifact components, [icaDecompose\(\)](#) for the built-in ICA implementation, [detectBadChannels\(\)](#) for channel-level artifact detection.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  colData = S4Vectors::DataFrame(label = paste0("Ch", 1:10)),
  samplingRate = 256
)

if (requireNamespace("fastICA", quietly = TRUE)) {
  ica_result <- runICA(pe, n_components = 5)
}
```

setAssaySamplingRate *Set a per-assay sampling rate (moved to PhysioCore)*

Description

Deprecated: delegates to `PhysioExperiment::setAssaySamplingRate()`. New code should use **PhysioExperiment** directly.

Usage

```
setAssaySamplingRate(x, assay_name, rate)
```

Arguments

x	A PhysioExperiment object.
assay_name	Name of the assay.
rate	Sampling rate for the assay in Hz.

Value

The updated PhysioExperiment.

See Also

[setAssaySamplingRate](#)

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(100 * 2), nrow = 100, ncol = 2)),
  samplingRate = 100
)
pe <- setAssaySamplingRate(pe, "raw", 100)
assaySamplingRates(pe)
```

sosDesign

Design a Butterworth filter in second-order-sections (SOS) form

Description

Builds the cascaded second-order-sections representation of a Butterworth filter directly from the analog prototype poles (bilinear transform), which is numerically stable for all orders. This is the design entry point for the causal / stateful filtering functions ([sosfilt](#), [sosfiltfilt](#), [StreamFilter](#)).

Usage

```
sosDesign(
  low = NULL,
  high = NULL,
  order = 4L,
  type = c("pass", "low", "high", "stop"),
  sr
)
```

Arguments

low	Lower cutoff frequency in Hz. Required for "high" and "pass"/"stop".
high	Upper cutoff frequency in Hz. Required for "low" and "pass"/"stop".
order	Filter order (per band). Default 4.
type	Filter type: "low", "high", "pass" (bandpass), or "stop" (bandstop).
sr	Sampling rate in Hz.

Value

A numeric matrix with one second-order section per row and six columns $c(b_0, b_1, b_2, a_0, a_1, a_2)$.

See Also

[sosfilt](#), [sosfiltfilt](#), [StreamFilter](#)

Examples

```
sos <- sosDesign(high = 40, type = "low", sr = 250, order = 4)
y <- sosfilt(sos, rnorm(1000))
```

 sosfilt

One-pass causal SOS filtering with optional carried state

Description

Filters x through a cascade of second-order sections, forward in time, mirroring `scipy.signal.sosfilt`. Each section uses the DF2T biquad recursion. Optionally carries the two-sample-per-section state so a signal can be processed in consecutive chunks identically to processing it whole.

Usage

```
sosfilt(sos, x, zi = NULL)
```

Arguments

<code>sos</code>	SOS matrix ($n_sections \times 6$), e.g. from sosDesign .
<code>x</code>	Numeric input vector.
<code>zi</code>	Optional initial state, a matrix $n_sections \times 2$. When supplied the return value is a list carrying the final state; when NULL (default) state starts at zero and a plain vector is returned. Use sosfiltInit for a transient-free warm start.

Value

If `zi` is NULL, the filtered numeric vector. Otherwise a list with `y` (filtered vector) and `zi` (final $n_sections \times 2$ state matrix).

See Also

[sosDesign](#), [sosfiltInit](#), [sosfiltfilt](#)

Examples

```
sos <- sosDesign(high = 40, type = "low", sr = 250, order = 4)
y <- sosfilt(sos, rnorm(1000))
# chunked with carried state == whole-signal filtering
x <- rnorm(1000)
zi0 <- matrix(0, nrow(sos), 2)
r1 <- sosfilt(sos, x[1:500], zi = zi0)
r2 <- sosfilt(sos, x[501:1000], zi = r1$zi)
max(abs(c(r1$y, r2$y) - sosfilt(sos, x)))
```

 sosfiltfilt

Zero-phase SOS filtering with state-initialised edge handling

Description

Forward-backward (zero-phase) SOS filtering that mirrors `scipy.signal.sosfiltfilt`: it odd-reflects the signal at both edges, initialises the delay-line to steady state (scaled by the boundary sample) to suppress start-up transients, filters forward and then backward. The result has zero phase distortion but is non-causal (both directions are used). For strictly causal / real-time-equivalent filtering, use [sosfilt](#) or [butterworthFilter](#) with `causal = TRUE`.

Usage

```
sosfiltfilt(sos, x, padlen = NULL)
```

Arguments

<code>sos</code>	SOS matrix (<code>n_sections</code> x 6).
<code>x</code>	Numeric input vector.
<code>padlen</code>	Number of samples of odd reflection padding at each edge. Defaults to $\min(3 * (2 * n_sections + 1), \text{length}(x) - 1)$.

Value

The zero-phase filtered numeric vector, same length as `x`.

See Also

[sosfilt](#), [sosDesign](#)

Examples

```
sos <- sosDesign(high = 40, type = "low", sr = 250, order = 4)
y <- sosfiltfilt(sos, rnorm(1000))
```

 sosfiltInit

Steady-state initial conditions for a cascaded SOS filter

Description

Per-section analogue of [lfilterInit](#) for second-order-sections form; equivalent to `scipy.signal.sosfilt_zi`. Each section's steady state is scaled by the DC gain of all preceding sections so that the whole cascade starts in steady state when the returned matrix (scaled by the first sample) is handed to [sosfilt](#).

Usage

```
sosfiltInit(sos)
```

Arguments

`sos` SOS matrix (n_sections x 6).

Value

A numeric matrix (n_sections x 2) of per-section initial states (unscaled).

See Also

[sosfilt](#), [lfilterInit](#)

Examples

```
sos <- sosDesign(high = 40, type = "low", sr = 250)
zi <- sosfiltInit(sos)
```

StreamFilter

Stateful streaming filter object

Description

Constructs a mutable filter object that carries its delay-line state across successive `$apply()` calls, so a signal fed in arbitrary chunks produces exactly the same output as filtering it whole. The identical math is used offline and online, which lets a real-time streaming backend reuse this object without re-deriving coefficients.

Usage

```
StreamFilter(sos = NULL, b = NULL, a = NULL, warmup = TRUE)
```

Arguments

<code>sos</code>	SOS matrix (n_sections x 6), e.g. from sosDesign . Provide either <code>sos</code> or both <code>b</code> and <code>a</code> .
<code>b, a</code>	Transfer-function coefficients (converted to a single SOS section only when they describe a biquad; otherwise supply <code>sos</code>).
<code>warmup</code>	Logical; if TRUE (default) the first <code>\$apply()</code> after construction or <code>\$reset()</code> initialises the state to steady state scaled by the first sample, suppressing the start-up transient. If FALSE, the state starts at zero.

Details

The returned object is an environment (reference semantics) with methods:

`$apply(x)` Filter numeric vector `x`, advancing and storing the internal state; returns the filtered vector.

`$reset()` Clear the carried state so the next `$apply()` starts fresh (cold or warm start per warmup).

`$state()` Return the current internal state matrix.

Value

An object of class "StreamFilter" (an environment).

See Also

[sosfilt](#), [sosDesign](#)

Examples

```
sos <- sosDesign(high = 30, type = "low", sr = 250, order = 4)
sf <- StreamFilter(sos)
x <- rnorm(1000)
y_chunked <- c(sf$apply(x[1:400]), sf$apply(x[401:1000]))
sf$reset()
y_whole <- sf$apply(x)
max(abs(y_chunked - y_whole))
```

zapLine

Remove line noise with ZapLine

Description

Removes power-line noise while preserving the broadband spectrum using ZapLine (de Cheveigne, 2020): the line is first suppressed by a comb-notch smoother, then a DSS/PCA of the notched-out residual isolates the line-noise spatial subspace, and only its top `nremove` components are removed from the data. This avoids the wide spectral notch of a fixed filter and needs at least two channels (line noise is treated as a spatial component).

Usage

```
zapLine(
  x,
  line_freq = 50,
  nremove = 1,
  nfft = NULL,
  assay_name = NULL,
  output_assay = "zapline"
)
```

Arguments

x	A PhysioExperiment object with a 2D (time x channels) assay.
line_freq	Line frequency in Hz (default: 50).
nremove	Number of line-noise components to remove (default: 1).
nfft	Optional analysis FFT length controlling the line-band width for the DSS bias (default: NULL = 1 Hz half-band).
assay_name	Input assay (default: defaultAssay(x)).
output_assay	Name for the cleaned assay (default: "zapline").

Value

The PhysioExperiment with the cleaned signal in output_assay; the removed line-power fraction is stored in metadata(x)\$line_noise and logged in provenance.

References

de Cheveigne, A. (2020). "ZapLine: A simple and effective method to remove power line artifacts." *NeuroImage*, 207, 116356. doi:[10.1016/j.neuroimage.2019.116356](https://doi.org/10.1016/j.neuroimage.2019.116356)

See Also

[cleanLine](#), [notchFilter](#) (the fast fixed notch)

Examples

```
pe <- PhysioExperiment(  
  assays = list(raw = matrix(rnorm(400 * 3), nrow = 400, ncol = 3)),  
  samplingRate = 200  
)  
pe <- zapLine(pe, line_freq = 50)
```

Index

applyPipeline, 3
applyPipeline(), 14, 29
asrCalibrate, 4, 5, 13
asrProcess, 4, 5, 13
assaySamplingRates, 6, 6

baselineCorrect, 6
baselineCorrect(), 31
butterworthFilter, 7, 18, 40
butterworthFilter(), 3, 14, 17–19, 21, 28, 34

classifyICAComponents, 9, 10
cleanData, 9
cleanLine, 11, 43
cleanRawdata, 4, 5, 12
createPipeline, 13
createPipeline(), 3, 29

decimate, 14
decimate(), 24, 35
detectArtifacts, 15
detectBadChannels, 10, 16
detectBadChannels(), 23, 32, 37
detrendSignal, 16
detrendSignal(), 18, 19
detrendSignals, 17
detrendSignals(), 14, 17, 31

filters-causal, 18
filterSignals, 19
filterSignals(), 14, 21, 28, 31
firFilter, 20
firFilter(), 19, 28

getCurrentReference, 21
getCurrentReference(), 26, 34

icaDecompose, 9, 10, 22
icaDecompose(), 37
icaRemove, 23
icaRemove(), 23, 32
interpolate, 24
interpolate(), 14, 35
interpolateBadChannels, 10, 25
isAverageReferenced, 25
isAverageReferenced(), 21, 34

lfilter, 26, 27
lfilter(), 20, 21
lfilterInit, 26, 27, 27, 40, 41

MultiPhysioExperiment, 6

notchFilter, 12, 28, 43
notchFilter(), 19, 21

print.PhysioPipeline, 29

rejectBadEpochs, 30
removeBaseline, 30
removeBaseline(), 18
removeICAComponents, 31
removeICAComponents(), 37
rereference, 32
rereference(), 21, 26
resample, 35
resample(), 3, 14, 24
runICA, 36
runICA(), 23, 32

setAssaySamplingRate, 37, 37
setAssaySamplingRate(), 35
signal::filtfilt(), 8, 20
sosDesign, 38, 39–42
sosfilt, 27, 38, 39, 40–42
sosfilt(), 8
sosfiltfilt, 38, 39, 40
sosfiltfiltInit, 27, 39, 40
sosfiltfiltInit(), 8
StreamFilter, 38, 41
StreamFilter(), 8

zapLine, [12](#), [42](#)