Package: PhysioPreprocess (via r-universe)

May 16, 2026

Title Preprocessing Functions for PhysioExperiment Objects

Version 0.2.0

Author Yusuke Matsui

Maintainer Yusuke Matsui <you@example.com>

Description Provides preprocessing functions for physiological signal data including digital filters (Butterworth, notch), resampling, artifact detection, ICA-based artifact removal, and preprocessing pipelines.

Depends R (>= 4.2), PhysioCore

Imports methods, SummarizedExperiment, S4Vectors, signal, stats

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

VignetteBuilder knitr

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

Collate 'filters.R' 'filters-advanced.R' 'resample.R' 'rereference.R' 'artifact.R' 'ica.R' 'detrending.R' 'preprocessing-pipeline.R' 'zzz.R'

RoxygenNote 7.3.3

Config/pak/sysreqs zlib1g-dev

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

Date/Publication 2026-03-16 11:31:04 UTC

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

RemoteRef HEAD

RemoteSha d97e7ae6e0fba1156692827b22f725177b8337e7

Contents

`.onLoad` *Package on-load hook*

Description

Package on-load hook

Usage

```
.onLoad(libname, pkg)
```

Arguments

`libname`	Library path.
`pkg`	Package name.

`applyPipeline` *Apply preprocessing pipeline*

Description

Applies a preprocessing pipeline to a PhysioExperiment object.

Usage

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

Arguments

`pe`	A PhysioExperiment object.
`pipeline`	A pipeline object created with createPipeline().
`verbose`	Logical. If TRUE, 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
)
## Not run:
pipeline <- createPipeline(
  detrend = list(fn = "detrendSignals", method = "linear")
)
pe_processed <- applyPipeline(pe, pipeline)

## End(Not run)
```

assaySamplingRates *Get sampling rates for all assays*

Description

Returns sampling rates associated with each assay. If per-assay rates have not been set, all assays are assumed to share the main sampling rate.

Usage

```
assaySamplingRates(x)
```

Arguments

x	A PhysioExperiment object.

Value

A named numeric vector where names correspond to assay names and values are their respective sampling rates in Hz.

References

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

See Also

`setAssaySamplingRate()` for setting per-assay rates, `resample()` for changing the sampling rate of data.

baselineCorrect *Baseline correction*

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 *Advanced signal filtering functions*

Description

This file provides advanced filtering operations including Butterworth, FIR, and notch filters for physiological signal processing. Butterworth filter

Usage

```
butterworthFilter(
  x,
  low = NULL,
  high = NULL,
  order = 4L,
  type = c("pass", "low", "high", "stop"),
  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).
output_assay	Name for the output assay. Default is "filtered".

Details

Applies a Butterworth filter (lowpass, highpass, bandpass, or bandstop) along the time axis of the specified assay. Uses zero-phase forward-backward filtering via signal::filtfilt() to avoid phase distortion.

Value

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

References

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

See Also

firFilter() for FIR filtering, notchFilter() for power line noise removal, filterSignals() for moving average filtering, detrendSignal() for trend removal.

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")
```

---

createPipeline *Preprocessing Pipeline for PhysioExperiment*

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 PhysioExperiment 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.

---

detectBadChannels *Detect bad channels*

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.

---

detrendSignal *Detrend signal*

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.

---

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 PhysioExperiment 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")
```

---

filterSignals *Moving average filter*

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

x	A PhysioExperiment object.
window	Integer window length for the moving average.
na.rm	Logical. If TRUE, NA values are ignored in the filter computation.
output_assay	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.

---

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",
  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".
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.

`getCurrentReference` *Get current reference*

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

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

Value

A list with four elements:

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

mixing The mixing matrix (channels x components).

unmixing The unmixing matrix (components x channels).

object The input `PhysioExperiment` with ICA components stored in the `"ica_components"` assay and ICA metadata.

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` *Remove ICA components*

Description

Removes specified ICA components and reconstructs the signal.

Usage

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

Arguments

`x`	A PhysioExperiment 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` *Interpolate signal*

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

`x`	A PhysioExperiment object.
`factor`	Integer interpolation factor.
`method`	Interpolation method: "linear" or "spline".
`output_assay`	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.

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

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

Value

Modified PhysioExperiment with interpolated channels.

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
```

---

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 `PhysioExperiment` 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 *Reject bad epochs*

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 *Remove baseline*

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 *Remove ICA components*

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

pe	A PhysioExperiment object.
ica_result	Result from runICA().
remove_components	Integer vector of component indices to remove.
assay_name	Name of the assay to reconstruct.
output_assay	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

```
## Not run:
pe <- runICA(pe)
pe_clean <- removeICAComponents(pe, ica_result, remove_components = c(1, 3))

## End(Not run)
```

---

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", "channel", "channels", "REST"),
  ref_channels = NULL,
  exclude = NULL,
  input_assay = NULL,
  output_assay = "rereferenced",
  keep_ref = TRUE
)
```

Arguments

x	A PhysioExperiment object.
ref_type	Type of re-referencing: "average" (common average reference), "channel" (single channel), "channels" (average of specified channels), or "REST" (Reference Electrode Standardization Technique).
ref_channels	For "channel" or "channels" type, the channel name(s) or index/indices to use as reference.
exclude	Channels to exclude from average reference calculation (e.g., non-EEG channels like EOG, EMG).
input_assay	Input assay name. If NULL, uses default assay.
output_assay	Output assay name. Default is "rereferenced".
keep_ref	Logical. If TRUE, keeps the original reference channel(s) in the output (zeroed). If FALSE, removes them.

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.

Value

A `PhysioExperiment` object with re-referenced data stored in a new assay named `output_assay`. The `reference` and `previous_reference` metadata fields are updated. When `keep_ref = FALSE` and using channel-based reference, the reference channel(s) are removed and a new object with reduced channel count is returned.

References

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

Changes the reference electrode for EEG recordings. Supports common re-referencing schemes including average reference, linked mastoids, and single electrode reference.

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

See Also

`getCurrentReference()` for querying the current reference, `isAverageReferenced()` for checking average reference status, `butterworthFilter()` for frequency-domain preprocessing.

Examples

```
# Create example EEG data
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  colData = S4Vectors::DataFrame(
    label = c("Fp1", "Fp2", "F3", "F4", "C3", "C4", "P3", "P4", "O1", "O2"),
    type = rep("EEG", 10)
  ),
  samplingRate = 256
)

# Apply average reference
pe_avg <- rereference(pe, ref_type = "average")

# Re-reference to a single channel (Cz)
pe_cz <- rereference(pe, ref_type = "channel", ref_channels = "C3")

# Re-reference to linked mastoids (if available)
# pe_linked <- rereference(pe, ref_type = "channels",
#                          ref_channels = c("M1", "M2"))
```

`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

`x`	A PhysioExperiment object.
`target_rate`	Target sampling rate in Hz.
`method`	Resampling method: "linear" (default), "spline", or "fft".
`assay_name`	Optional assay name. If NULL, uses the default assay.
`output_assay`	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.

`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
)
## Not run:
ica_result <- runICA(pe, n_components = 5)

## End(Not run)
```

---

setAssaySamplingRate *Set sampling rate for a specific assay*

Description

Records a per-assay sampling rate in the object metadata. Useful when different assays have been resampled to different rates.

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

A PhysioExperiment object with updated per-assay sampling rate metadata.

References

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

See Also

assaySamplingRates() for retrieving per-assay rates, resample() for changing the sampling rate of data.

Index