

Package: dynR (via r-universe)

October 5, 2026

Title Dynamic Connectivity Analysis for Neurophysiological Timeseries

Version 0.1.5

Description An R port of the Python dynfc library for computing dynamic connectivity (dynFC) representations from multivariate neurophysiological timeseries, including BOLD fMRI, EEG, LFP, and related signals. Implements sliding-window Pearson correlation (Hansen et al., 2015), edge-centric co-fluctuation analysis (Esfahlani et al., 2020; Faskowitz et al., 2020), instantaneous phase-locking via the Hilbert transform, dynamic phase-locking matrices (dPL), the LEiDA leading-eigenvector framework (Cabral et al., 2017; Lord et al., 2019), and the Kuramoto order parameter with metastability and Shannon entropy. Part of the Circadia Lab R ecosystem.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Depends R (>= 4.1.0)

Imports dplyr (>= 1.1.0), tidyr (>= 1.3.0), rlang (>= 1.1.0), Rcpp (>= 1.0.0), ggplot2 (>= 3.4.0)

LinkingTo Rcpp

Suggests testthat (>= 3.0.0), knitr, rmarkdown, pkgdown, withr, reticulate

Config/testthat/edition 3

Lifecycle experimental

URL <https://dynr.circadia-lab.uk>, <https://github.com/circadia-bio/dynR>

BugReports <https://github.com/circadia-bio/dynR/issues>

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

Config/pak/sysreqs libicu-dev
Repository <https://circadia-bio.r-universe.dev>
Date/Publication 2026-07-06 21:02:48 UTC
RemoteUrl <https://github.com/circadia-bio/dynR>
RemoteRef v0.1.5
RemoteSha 3a50ec7e3e97110739147976c98daf0f5be07eb8

Contents

bandpass_filter	2
batch_leida	3
batch_sw	4
cofluct	5
corr_corr	6
corr_slide	7
do_euclid	8
dyn_phase_lock	8
dyn_transitions	9
fc	11
get_leida	11
hilbert_phases	12
kuramoto	13
leida_pipeline	14
plot.dynR_leida	15
plot.dynR_sw	16
plot_fc	17
plot_state_sequence	17
plot_synchrony	18
shannon_entropy	19
stack_leida	19
stack_synchrony	20
sw_pipeline	21
ts	22

Index	24
--------------	-----------

bandpass_filter	<i>Butterworth bandpass filter</i>
-----------------	------------------------------------

Description

Design and apply a zero-phase Butterworth bandpass filter to a signal, using scipy-compatible steady-state initial conditions (equivalent to `scipy.signal.sosfiltfilt/scipy.signal.filtfilt` with `padtype = "odd"`).

Usage

```
bandpass_filter(x, flp, fhi, delt, order = 2)
```

Arguments

x	Numeric vector. Signal to be filtered.
flp	Numeric. Low-pass cutoff frequency (Hz).
fhi	Numeric. High-pass cutoff frequency (Hz).
delt	Numeric. Sampling interval in seconds (i.e. TR for fMRI).
order	Integer. Filter order. Default is 2.

Details

Two internal code paths are used depending on order:

- order <= 3 (typical fMRI use): direct-form b/a coefficients with the companion-matrix initial-condition method. `scipy.filtfilt`-parity validated to < 1e-9 for standard fMRI parameters.
- order >= 4: second-order sections (SOS). The companion matrix for the full-order b/a representation becomes ill-conditioned for high-order filters with poles close to $z = 1$ (very low lower cutoff). SOS avoids this by decomposing the filter into biquad sections, each with a 2x2 companion matrix that is always well-conditioned.

Filter coefficients are computed by `.butter_bandpass_zpk()`, a bespoke port of `scipy.signal.butter()` that requires no external packages.

Value

Numeric vector. Zero-phase filtered signal, same length as x.

Examples

```
set.seed(1)
x <- rnorm(200)
x_filt <- bandpass_filter(x, flp = 0.01, fhi = 0.1, delt = 2, order = 2)
```

batch_leida

Batch LEiDA pipeline across subjects

Description

Apply `leida_pipeline()` to multiple subjects in one call. Accepts either a named list of [N x Tmax] matrices or a 3-D array [N x Tmax x subjects].

Usage

```
batch_leida(data, ...)
```

Arguments

`data` Named list of [N x Tmax] numeric matrices, **or** a 3-D numeric array with dimensions [N, Tmax, subjects]. For arrays, subject names are taken from `dimnames(data)[[3]]` if set, otherwise "sub1", "sub2", etc.

`...` Additional arguments passed to `leida_pipeline()` (e.g. `flp`, `fhi`, `delt`, `filter`).

Value

Named list of `dynR_leida` objects, one per subject.

See Also

`stack_leida()`, `leida_pipeline()`

Examples

```
set.seed(1)
ts_list <- list(
  sub01 = matrix(rnorm(10 * 200), nrow = 10),
  sub02 = matrix(rnorm(10 * 200), nrow = 10)
)
results <- batch_leida(ts_list, filter = FALSE)
names(results)
```

batch_sw

Batch sliding-window pipeline across subjects

Description

Apply `sw_pipeline()` to multiple subjects in one call. Accepts either a named list of [N x Tmax] matrices or a 3-D array [N x Tmax x subjects].

Usage

```
batch_sw(data, window, ...)
```

Arguments

`data` Named list of [N x Tmax] matrices, or a 3-D array [N, Tmax, subjects]. See `batch_leida()`.

`window` Integer. Window size in timepoints passed to `sw_pipeline()`.

`...` Additional arguments passed to `sw_pipeline()`.

Value

Named list of `dynR_sw` objects, one per subject.

See Also

batch_leida(), sw_pipeline()

Examples

```
set.seed(2)
ts_list <- list(
  sub01 = matrix(rnorm(10 * 200), nrow = 10),
  sub02 = matrix(rnorm(10 * 200), nrow = 10)
)
results <- batch_sw(ts_list, window = 20, filter = FALSE)
names(results)
```

cofluct

Edge-centric cofluctuation analysis

Description

Compute edge time series and root-sum-square (RSS) cofluctuations from z-standardised BOLD signal. Implements the edge-centric FC framework of Esfahlani et al. (2020) and Faskowitz et al. (2020).

Usage

```
cofluct(timeseries, k = 1)
```

Arguments

timeseries	Numeric matrix [$N \times T_{\max}$]. BOLD signal with N parcels as rows and $T_{\max}$ timepoints as columns.
k	Integer. Upper-triangle offset. $k = 1$ (default) excludes the diagonal; $k = 0$ includes it.

Value

A list with:

edge_ts	Numeric matrix [$n_{\text{edges}} \times T_{\max}$]. Edge time series, one row per unique parcel pair.
rss	Numeric vector [$T_{\max}$]. Root-sum-square cofluctuation at each timepoint.

References

Esfahlani, F. Z. et al. (2020). High-amplitude cofluctuations in cortical activity drive functional connectivity. *PNAS*, 117(45), 28393–28401. doi:[10.1073/pnas.2005531117](https://doi.org/10.1073/pnas.2005531117)

Faskowitz, J. et al. (2020). Edge-centric functional network representations of human cerebral cortex reveal overlapping system-level architecture. *Nature Neuroscience*, 23(12), 1644–1654. doi:[10.1038/s41593-020-00719-y](https://doi.org/10.1038/s41593-020-00719-y)

Examples

```
set.seed(1)
ts <- matrix(rnorm(10 * 200), nrow = 10, ncol = 200)
res <- cofluct(ts)
dim(res$edge_ts) # n_edges x 200
```

corr_corr	<i>Correlation of correlations matrix</i>
-----------	---

Description

Compute the correlation between edge time series across all timepoints, producing a $[T_{\max} \times T_{\max}]$ "correlation of correlations" matrix (Hansen et al., 2015).

Usage

```
corr_corr(timeseries, k = 1)
```

Arguments

timeseries	Numeric matrix $[N \times T_{\max}]$. BOLD signal.
k	Integer. Upper-triangle offset passed to <code>cofluct()</code> . Default 1.

Value

Numeric matrix $[T_{\max} \times T_{\max}]$.

References

Hansen, E. C. A. et al. (2015). Functional connectivity dynamics: Modeling the switching behavior of the resting state. *NeuroImage*, 105, 525–535. doi:[10.1016/j.neuroimage.2014.11.001](https://doi.org/10.1016/j.neuroimage.2014.11.001)

Examples

```
set.seed(1)
ts <- matrix(rnorm(10 * 100), nrow = 10, ncol = 100)
cc <- corr_corr(ts)
dim(cc) # 100 x 100
```

corr_slide	<i>Sliding window correlation</i>
------------	-----------------------------------

Description

Compute functional connectivity matrices over sliding windows of a BOLD timeseries (Hansen et al., 2015).

Usage

```
corr_slide(timeseries, window, step = NULL)
```

Arguments

timeseries	Numeric matrix [N x Tmax]. BOLD signal with N parcels as rows and Tmax timepoints as columns.
window	Integer. Window size in timepoints.
step	Integer. Step between window onsets. Defaults to window (non-overlapping windows).

Details

The correlation matrices are computed by a compiled C++ backend (`corr_slide_cpp()`) that calculates Pearson correlation directly on the input matrix. No per-window submatrix allocation or transposition is performed; the t-outer loop ordering exploits the column-major memory layout of the input for sequential cache access.

Value

A list with:

corr_mats	Array [N, N, n_windows] of Pearson correlation matrices.
idx	Integer vector of 1-indexed window onset positions.

References

Hansen, E. C. A. et al. (2015). Functional connectivity dynamics: Modeling the switching behavior of the resting state. *NeuroImage*, 105, 525-535. doi:[10.1016/j.neuroimage.2014.11.001](https://doi.org/10.1016/j.neuroimage.2014.11.001)

Examples

```
set.seed(1)
ts <- matrix(rnorm(10 * 200), nrow = 10, ncol = 200)
res <- corr_slide(ts, window = 20)
dim(res$corr_mats) # 10 x 10 x 10
```

do_euclid	<i>Euclidean distance between consecutive points</i>
-----------	--

Description

Compute the Euclidean distance between each consecutive pair of rows in a matrix — typically a trajectory through PC space.

Usage

```
do_euclid(x)
```

Arguments

`x` Numeric matrix [`n_points` $\times$ `n_dims`].

Value

Numeric vector of length `nrow(x)`. The first element is always 0 (no previous point).

Examples

```
set.seed(1)
pcs <- matrix(rnorm(50 * 3), nrow = 50, ncol = 3)
d <- do_euclid(pcs)
length(d) # 50
```

dyn_phase_lock	<i>Dynamic phase-locking matrix (dPL)</i>
----------------	---

Description

Compute instantaneous phase-locking matrices from parcel-level phase time series and extract leading eigenvectors via [get_leida\(\)](#). The first and last 10 timepoints are discarded to avoid edge effects from the Hilbert transform.

Usage

```
dyn_phase_lock(phases)
```

Arguments

`phases` Numeric matrix [`N` $\times$ `Tmax`]. Instantaneous phases in radians, as returned by [hilbert_phases\(\)](#).

Details

The phase-locking computation is accelerated by a compiled C++ backend (`dyn_phase_lock_cpp()`): only the upper triangle is evaluated with $\cos(\phi_i - \phi_j)$ and mirrored to the lower, halving trigonometric operations relative to a naive double loop.

Value

A list with:

<code>sync_conn</code>	Array [N, N, Tmax-20]. Instantaneous phase-locking matrices, one per (trimmed) timepoint.
<code>leida</code>	Matrix [Tmax-20, N]. Leading eigenvectors from <code>get_leida()</code> .

References

- Cabral, J. et al. (2017). Cognitive performance in healthy older adults relates to spontaneous switching between states of functional connectivity during rest. *Scientific Reports*, 7(1), 5135. [doi:10.1038/s41598017054257](https://doi.org/10.1038/s41598017054257)
- Lord, L.-D. et al. (2019). Dynamical exploration of the repertoire of brain networks at rest is modulated by psilocybin. *NeuroImage*, 199, 127-142. [doi:10.1016/j.neuroimage.2019.05.060](https://doi.org/10.1016/j.neuroimage.2019.05.060)

Examples

```
set.seed(1)
ts <- matrix(rnorm(10 * 200), nrow = 10, ncol = 200)
phases <- hilbert_phases(ts)
res <- dyn_phase_lock(phases)
dim(res$sync_conn) # 10 x 10 x 180
dim(res$leida)     # 180 x 10
```

<code>dyn_transitions</code>	<i>State transition probabilities (Markov analysis)</i>
------------------------------	---

Description

From a long-format data frame of cluster labels ordered in time, compute first-order Markov transition probabilities between brain states: for each source state, the fraction of transitions that lead to each target state.

Usage

```
dyn_transitions(tbl, vars, cVar, sortBy, groupBy, remIntra = FALSE)
```

Arguments

tbl	A data frame in long format (one row per timepoint).
vars	Character vector of grouping / covariate column names to preserve in the output (e.g. <code>c("sub", "ses", "age", "sex")</code>).
cVar	Character. Name of the column holding integer cluster labels.
sortBy	Character vector of column names used to sort rows within each group before transitions are computed (typically <code>c("sub", "ses", "ttime")</code>).
groupBy	Character vector of column names that define independent sequences (typically <code>c("sub", "ses")</code>). Transitions are never computed across group boundaries.
remIntra	Logical. If TRUE, self-transitions (state $\rightarrow$ same state) are removed before normalising probabilities. Default FALSE.

Details

Ported from `clusters_markov()` in the `neonatal_dfc` analysis pipeline (França et al., *Nat Commun*).

Value

A nested tibble with columns `tag`, `source`, `target`, and `data`. `tag` encodes the transition as "`<source>_<target>`". Each data element is a per-group tibble with columns inherited from `vars` plus:

n	Raw transition count.
tot	Total transitions out of source for that group.
nCount	Transition probability (n / tot).

Examples

```
set.seed(1)
df <- data.frame(
  sub   = rep(c("A", "B"), each = 50),
  ses   = 1L,
  ttime = rep(seq_len(50), 2),
  clus4 = sample(1:4, 100, replace = TRUE)
)
tr <- dyn_transitions(
  df,
  vars   = c("sub", "ses"),
  cVar    = "clus4",
  sortBy = c("sub", "ses", "ttime"),
  groupBy = c("sub", "ses")
)
tr
```

fc	<i>Functional connectivity matrix (200 parcels)</i>
----	---

Description

Static functional connectivity matrix derived from [ts](#) by computing the full-timeseries Pearson correlation across all 200 parcel pairs. Serves as a ground-truth reference for validating sliding-window and phase-based dynFC methods when the window spans the full timeseries.

Usage

fc

Format

A numeric matrix with 200 rows and 200 columns. Diagonal is 1; off-diagonal values are Pearson correlations in [-1, 1].

Source

Derived from [ts](#).

get_leida	<i>Leading eigenvector decomposition (LEiDA)</i>
-----------	--

Description

Extract the leading eigenvector from a series of instantaneous phase-locking matrices. The sign of each eigenvector is normalised so that its sum is non-positive, following the LEiDA convention.

Usage

get_leida(sync_conn)

Arguments

sync_conn	Numeric array [N, N, Tmax]. Instantaneous phase-locking matrices, as returned by dyn_phase_lock() .
-----------	---

Details

The computation is accelerated by a compiled C++ / LAPACK backend (`get_leida_cpp()`): LAPACK dsyev is called once per timepoint with a single shared workspace allocation, and only the leading eigenvector (last column of the ascending-order output) is retained.

Value

Numeric matrix [Tmax x N]. Leading eigenvectors, one per timepoint.

References

Cabral, J. et al. (2017). Cognitive performance in healthy older adults relates to spontaneous switching between states of functional connectivity during rest. *Scientific Reports*, 7(1), 5135. doi:10.1038/s41598017054257

Lord, L.-D. et al. (2019). Dynamical exploration of the repertoire of brain networks at rest is modulated by psilocybin. *NeuroImage*, 199, 127-142. doi:10.1016/j.neuroimage.2019.05.060

hilbert_phases

Hilbert transform phase extraction

Description

Compute the instantaneous phase time series for all parcels/voxels via the analytic signal (Hilbert transform). Each parcel timeseries is demeaned before transformation.

Usage

```
hilbert_phases(timeseries)
```

Arguments

`timeseries` Numeric matrix [N x Tmax]. BOLD signal with N parcels as rows and Tmax timepoints as columns.

Details

All N parcels are processed in two vectorised calls to `mvfft()` rather than an N-iteration R loop over `fft()`: the timeseries matrix is transposed to a Tmax x N layout so `mvfft` applies the FFT to each parcel column simultaneously, the Hilbert multiplier is broadcast across columns, and the inverse FFT recovers the analytic signal for all parcels at once.

`.hilbert_r()` is retained as an internal single-vector reference (scipy-parity validated).

Value

Numeric matrix [N x Tmax]. Instantaneous phases in radians.

References

Cabral, J. et al. (2017). Cognitive performance in healthy older adults relates to spontaneous switching between states of functional connectivity during rest. *Scientific Reports*, 7(1), 5135. doi:10.1038/s41598017054257

Examples

```
set.seed(1)
ts <- matrix(rnorm(10 * 200), nrow = 10, ncol = 200)
phases <- hilbert_phases(ts)
dim(phases) # 10 x 200
```

kuramoto

Kuramoto order parameter and metastability

Description

Compute the global Kuramoto order parameter time series, the metastability index (standard deviation of the order parameter), and Shannon entropy of synchrony from parcel-level instantaneous phase time series. The first and last 10 timepoints are discarded (matching [dyn_phase_lock\(\)](#)).

Usage

```
kuramoto(phases, base = 2, n_bits = 8L)
```

Arguments

phases	Numeric matrix [N x Tmax]. Instantaneous phases in radians, as returned by hilbert_phases() .
base	Numeric. Logarithm base for Shannon entropy. Default is 2.
n_bits	Integer or NULL. Bit depth for discretising the synchrony series before entropy estimation. Default is 8.

Details

The synchrony time series is computed by a compiled C++ backend ([kuramoto_sync_cpp\(\)](#)) that accumulates cos and sin components directly, avoiding R complex-number allocation and vapply dispatch overhead.

Value

A list with:

metastability	Numeric scalar. Standard deviation of the Kuramoto order parameter.
synchrony	Numeric vector [Tmax-20]. Kuramoto order parameter at each (trimmed) time-point.
entropy	Numeric scalar. Shannon entropy of the synchrony series.

Examples

```
set.seed(1)
ts <- matrix(rnorm(10 * 200), nrow = 10, ncol = 200)
phases <- hilbert_phases(ts)
res <- kuramoto(phases)
res$metastability
```

leida_pipeline	<i>LEiDA pipeline</i>
----------------	-----------------------

Description

Convenience wrapper running the full phase-based dynamic FC pipeline: optional Butterworth bandpass filter -> Hilbert transform -> dynamic phase-locking matrices + LEiDA eigenvectors -> Kuramoto order parameter. Returns a structured dynR_leida object with print() and plot() methods.

Usage

```
leida_pipeline(
  timeseries,
  flp = NULL,
  fhi = NULL,
  delt = NULL,
  order = 2L,
  filter = TRUE
)
```

Arguments

timeseries	Numeric matrix [N x Tmax]. Rows = channels/parcels, columns = timepoints.
flp	Numeric. Low-pass cutoff frequency (Hz). No default — must be set explicitly when filter = TRUE.
fhi	Numeric. High-pass cutoff frequency (Hz). No default — must be set explicitly when filter = TRUE.
delt	Numeric. Sampling interval in seconds (1 / sampling_rate). No default — must be set explicitly when filter = TRUE.
order	Integer. Butterworth filter order. Default 2L.
filter	Logical. Apply bandpass filter before phase extraction? Default TRUE. Set to FALSE if the timeseries is already filtered.

Details

dynR is modality-agnostic. flp, fhi, and delt have no defaults and **must** be supplied when filter = TRUE. Set them for your recording modality (see examples). Pass filter = FALSE if the timeseries is already band-limited.

Value

An object of class dynR_leida (a named list) with elements:

leida	Matrix [Tmax-20, N]. LEiDA eigenvectors.
sync_conn	Array [N, N, Tmax-20]. Instantaneous phase-locking matrices.

phases	Matrix [N, Tmax]. Instantaneous phases in radians.
synchrony	Numeric vector [Tmax-20]. Kuramoto R(t).
metastability	Numeric. Standard deviation of synchrony.
entropy	Numeric. Shannon entropy of synchrony series (bits).
N	Integer. Number of channels/parcels.
Tmax	Integer. Number of timepoints.

See Also

sw_pipeline(), dyn_phase_lock(), kuramoto(), plot.dynR_leida()

Examples

```
set.seed(1)
ts <- matrix(rnorm(10 * 200), nrow = 10)

# filter = FALSE: timeseries already band-limited
res <- leida_pipeline(ts, filter = FALSE)
res

# fMRI BOLD (TR = 2 s)
## Not run:
res <- leida_pipeline(ts, flp = 0.01, fhi = 0.1, delt = 2)

## End(Not run)

# EEG alpha band (250 Hz)
## Not run:
res <- leida_pipeline(ts, flp = 8, fhi = 12, delt = 1 / 250)

## End(Not run)
```

plot.dynR_leida	<i>Plot method for dynR_leida objects</i>
-----------------	---

Description

Plot method for dynR_leida objects

Usage

```
## S3 method for class 'dynR_leida'
plot(x, type = c("synchrony", "fc"), ...)
```

Arguments

x	A dynR_leida object as returned by leida_pipeline().
type	Character. "synchrony" (default) plots the Kuramoto order parameter time series via plot_synchrony(); "fc" plots the time-averaged phase-locking matrix via plot_fc().
...	Additional arguments passed to the underlying plot function.

Value

A ggplot object.

plot.dynR_sw	<i>Plot method for dynR_sw objects</i>
--------------	--

Description

Plot method for dynR_sw objects

Usage

```
## S3 method for class 'dynR_sw'
plot(x, type = c("rss", "fc"), ...)
```

Arguments

x	A dynR_sw object as returned by sw_pipeline().
type	Character. "rss" (default) plots the RSS cofluctuation time series; "fc" plots the time-averaged sliding-window FC matrix via plot_fc().
...	Additional arguments passed to the underlying plot function.

Value

A ggplot object.

plot_fc	<i>Plot a functional connectivity matrix</i>
---------	--

Description

Render a square correlation or phase-locking matrix as a heatmap using the dynR diverging palette (deep indigo -> periwinkle -> brick red), centred at zero. Matches the colour scheme used in the dynR vignettes and pkgdown site.

Usage

```
plot_fc(fc_matrix, title = "Functional connectivity", limits = c(-1, 1))
```

Arguments

fc_matrix	Numeric matrix [N x N]. Pearson correlations or phase-locking values.
title	Character. Plot title. Default "Functional connectivity".
limits	Numeric vector c(lo, hi). Colour scale limits. Default c(-1, 1).

Value

A ggplot object.

Examples

```
data(fc, package = "dynR")
plot_fc(fc[1:20, 1:20])
```

plot_state_sequence	<i>Plot a brain state sequence</i>
---------------------	------------------------------------

Description

Tile plot of brain state labels over time (or windows), coloured with the dynR state palette (periwinkle, brick red, mauve, deep indigo, dusty rose).

Usage

```
plot_state_sequence(
  states,
  x_label = "Timepoint",
  title = "Brain state sequence",
  palette = NULL
)
```

Arguments

states	Integer or factor vector of state labels.
x_label	Character. x-axis label. Default "Timepoint".
title	Character. Plot title. Default "Brain state sequence".
palette	Character vector of colours. Defaults to the dynR 5-colour state palette, recycled if $K > 5$.

Value

A ggplot object.

Examples

```
set.seed(1)
states <- sample(1:4, 100, replace = TRUE)
plot_state_sequence(states)
```

plot_synchrony	<i>Plot Kuramoto synchrony time series</i>
----------------	--

Description

Line plot of the Kuramoto order parameter $R(t)$ over time, with a dashed horizontal line at the mean. Uses the dynR periwinkle/brick-red palette.

Usage

```
plot_synchrony(synchrony, title = "Kuramoto order parameter")
```

Arguments

synchrony	Numeric vector. Kuramoto $R(t)$, as returned by kuramoto() or the \$synchrony element of leida_pipeline().
title	Character. Plot title.

Value

A ggplot object.

Examples

```
set.seed(1)
ts <- matrix(rnorm(10 * 200), nrow = 10)
ph <- hilbert_phases(ts)
kop <- kuramoto(ph)
plot_synchrony(kop$synchrony)
```

shannon_entropy	<i>Shannon entropy</i>
-----------------	------------------------

Description

Estimate Shannon entropy from a numeric vector, with optional bit-depth discretisation (as used internally by [kuramoto\(\)](#)).

Usage

```
shannon_entropy(x, base = 2, n_bits = NULL)
```

Arguments

x	Numeric vector.
base	Numeric. Logarithm base. Default is 2 (entropy in bits).
n_bits	Integer or NULL. If supplied, values are scaled by 2^{n_bits} and rounded before computing entropy. Default is NULL.

Value

Numeric scalar. Shannon entropy.

Examples

```
x <- sample(1:4, 100, replace = TRUE)
shannon_entropy(x)
```

stack_leida	<i>Stack LEiDA eigenvectors across subjects</i>
-------------	---

Description

Combine the LEiDA eigenvector matrices from a list of dynR_leida results into a single matrix (or data frame) ready for cross-subject K-means clustering.

Usage

```
stack_leida(batch_result, add_subject_id = TRUE)
```

Arguments

batch_result	Named list of dynR_leida objects, as returned by batch_leida().
add_subject_id	Logical. Prepend a subject column with the subject name? Default TRUE.

Value

If `add_subject_id = TRUE`: a data frame with columns `subject` and one column per parcel (named `V1`, `V2`, ...). If `FALSE`: a plain numeric matrix [`total_timepoints` x `N`].

See Also

`batch_leida()`

Examples

```
set.seed(3)
ts_list <- list(
  sub01 = matrix(rnorm(10 * 200), nrow = 10),
  sub02 = matrix(rnorm(10 * 200), nrow = 10)
)
results <- batch_leida(ts_list, filter = FALSE)
leida_all <- stack_leida(results)
dim(leida_all)
```

stack_synchrony

Stack synchrony vectors across subjects

Description

Combine the Kuramoto synchrony time series from a list of `dynR_leida` results into a tidy data frame with one row per timepoint per subject.

Usage

```
stack_synchrony(batch_result)
```

Arguments

`batch_result` Named list of `dynR_leida` objects.

Value

A data frame with columns:

<code>subject</code>	Subject identifier.
<code>timepoint</code>	Timepoint index (within subject).
<code>synchrony</code>	Kuramoto $R(t)$.
<code>metastability</code>	Subject-level metastability (constant within subject).

See Also

`batch_leida()`

Examples

```
set.seed(4)
ts_list <- list(
  sub01 = matrix(rnorm(10 * 200), nrow = 10),
  sub02 = matrix(rnorm(10 * 200), nrow = 10)
)
results <- batch_leida(ts_list, filter = FALSE)
df_sync <- stack_synchrony(results)
head(df_sync)
```

sw_pipeline

Sliding-window pipeline

Description

Convenience wrapper running the full correlation-based dynamic FC pipeline: optional Butterworth bandpass filter -> sliding-window Pearson correlation matrices -> edge-centric co-fluctuations. Returns a structured dynR_sw object with `print()` and `plot()` methods.

Usage

```
sw_pipeline(
  timeseries,
  window,
  step = NULL,
  flp = NULL,
  fhi = NULL,
  delt = NULL,
  order = 2L,
  filter = TRUE
)
```

Arguments

timeseries	Numeric matrix [N x Tmax].
window	Integer. Window size in timepoints.
step	Integer. Step between window onsets. Default: window (non-overlapping windows).
flp	Numeric. Low-pass cutoff (Hz). No default — required when filter = TRUE.
fhi	Numeric. High-pass cutoff (Hz). No default — required when filter = TRUE.
delt	Numeric. Sampling interval in seconds. No default — required when filter = TRUE.
order	Integer. Butterworth filter order. Default 2L.
filter	Logical. Apply bandpass filter? Default TRUE.

Details

dynR is modality-agnostic. `flp`, `fhi`, and `delt` have no defaults and **must** be supplied when `filter = TRUE`. Pass `filter = FALSE` if the timeseries is already band-limited.

Value

An object of class `dynR_sw` (a named list) with elements:

<code>corr_mats</code>	Array [N, N, n_windows]. Sliding FC matrices.
<code>idx</code>	Integer vector. 1-indexed window onset positions.
<code>edge_ts</code>	Matrix [n_edges, Tmax]. Edge time series.
<code>rss</code>	Numeric vector [Tmax]. Root-sum-square co fluctuation.
<code>window</code>	Integer. Window size used.
<code>step</code>	Integer. Step size used.
<code>N</code>	Integer. Number of channels/parcels.
<code>Tmax</code>	Integer. Number of timepoints.

See Also

`leida_pipeline()`, `corr_slide()`, `cofluct()`, `plot.dynR_sw()`

Examples

```
set.seed(1)
ts <- matrix(rnorm(10 * 200), nrow = 10)

# filter = FALSE: timeseries already band-limited
res <- sw_pipeline(ts, window = 20, filter = FALSE)
res

# fMRI BOLD (TR = 2 s)
## Not run:
res <- sw_pipeline(ts, window = 20, flp = 0.01, fhi = 0.1, delt = 2)

## End(Not run)
```

ts

BOLD fMRI timeseries (200 parcels, 600 timepoints)

Description

A resting-state BOLD fMRI timeseries from the [edge-ts](#) repository, parcellated into 200 regions of interest. Used in dynR vignettes and examples to demonstrate dynamic FC methods on realistic data.

Usage

ts

Format

A numeric matrix with 200 rows (parcels) and 600 columns (timepoints).

Source

<https://github.com/brain-networks/edge-ts>

Index

- * **datasets**
 - fc, [11](#)
 - ts, [22](#)
- bandpass_filter, [2](#)
- batch_leida, [3](#)
- batch_sw, [4](#)
- cofluct, [5](#)
- cofluct(), [6](#)
- corr_corr, [6](#)
- corr_slide, [7](#)
- do_euclid, [8](#)
- dyn_phase_lock, [8](#)
- dyn_phase_lock(), [11](#), [13](#)
- dyn_transitions, [9](#)
- fc, [11](#)
- fft(), [12](#)
- get_leida, [11](#)
- get_leida(), [8](#), [9](#)
- hilbert_phases, [12](#)
- hilbert_phases(), [8](#), [13](#)
- kuramoto, [13](#)
- kuramoto(), [19](#)
- leida_pipeline, [14](#)
- mvfft(), [12](#)
- plot.dynR_leida, [15](#)
- plot.dynR_sw, [16](#)
- plot_fc, [17](#)
- plot_state_sequence, [17](#)
- plot_synchrony, [18](#)
- shannon_entropy, [19](#)
- stack_leida, [19](#)
- stack_synchrony, [20](#)
- sw_pipeline, [21](#)
- ts, [11](#), [22](#)