

Package: neurocluster (via r-universe)

October 16, 2026

Type Package

Title Spatially Constrained Clustering for Neuroimaging Data

Version 0.1.0.9000

Description Provides multiple algorithms for spatially constrained clustering of 3D and 4D neuroimaging data. Methods include supervoxels, SNIC (Simple Non-Iterative Clustering), SLIC (Simple Linear Iterative Clustering), slice-based minimum spanning forest, FLASH-3D with DCT compression, and adaptive correlation superclustering (ACSC). All methods balance feature similarity and spatial proximity to produce coherent brain parcellations. A unified 'cluster4d' interface standardizes access to all algorithms.

License GPL (>= 3)

URL <https://bbuchsbaum.github.io/neurocluster/>,
<https://github.com/bbuchsbaum/neurocluster>

BugReports <https://github.com/bbuchsbaum/neurocluster/issues>

Encoding UTF-8

Depends R (>= 4.1)

Imports Rcpp (>= 0.11.3), RcppParallel, Matrix, FNN, RANN, assertthat, clue, igraph, adjoin, neuroim2, purrr, future, future.apply, furr, methods, eigencore (>= 1.0.3)

LinkingTo Rcpp, RcppParallel, RcppArmadillo

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE)

Suggests testthat, knitr, rmarkdown, pkgdown, ggplot2, rsvd, neurosurf, irlba, robustbase, mclust, albersdown, processx

VignetteBuilder knitr

Remotes bbuchsbaum/neurosurf

Config/Needs/website albersdown

Config/pak/sysreqs cmake libglpk-dev libicu-dev libxml2-dev zlib1g-dev

Repository <https://bbuchsbaum.r-universe.dev>
Date/Publication 2026-10-16 15:55:08 UTC
RemoteUrl <https://github.com/bbuchsbaum/neurocluster>
RemoteRef HEAD
RemoteSha 8b03000e14518f9cc21a874c995d2fbd3ba305f3

Contents

acsc	3
check_cluster_ordering	5
cl_class_ids.cluster_result	6
cluster_metrics	7
cluster4d	8
cluster4d_acsc	12
cluster4d_brsslic	13
cluster4d_commute	16
cluster4d_corrslc	17
cluster4d_flash3d	20
cluster4d_g3s	21
cluster4d_mcl	22
cluster4d_rena	24
cluster4d_rena_plus	25
cluster4d_slic	26
cluster4d_slice_msf	28
cluster4d_snic	30
cluster4d_supervoxels	31
clustering_accuracy	33
commute_cluster	34
compare_cluster4d	35
compress_features_svd	36
compute_centroids	39
compute_functional_gradient	40
detrend_basis_cpp	41
detrend_dct_cpp	42
detrend_poly_cpp	43
detrend_time	43
detrend_time_cpp	45
er_ml_rena	45
estimate_optimal_components	46
find_gradient_seeds_g3s	47
find_initial_points	49
generate_synthetic_volume	50
is.cl_partition.cluster_result	51
knn_shrink	52
make_block_synthetic	53
make_dct_basis	53

make_poly_basis	54
make_synthetic_clusters	55
merge_clus	56
merge_clus.cluster_result	57
meta_clust	58
meta_clust.cluster_result	58
neurocluster_simd_info	60
normalize_detrend	61
normalize_detrend_cpp	62
normalize_volumes	62
normalize_volumes_cpp	63
plot.cluster4d_result	64
print.cluster4d_result	64
print.g3s_result	65
randomize_cluster_ids	65
rena	67
rena_plus	70
slic4d_grad_summary	71
slic4d_supervoxels	72
slice_msf	74
slice_msf_consensus	76
slice_msf_single	77
snic	78
spatial_gradient	81
suggest_cluster4d_params	82
summary.cluster4d_result	83
supervoxel_cluster_surface	83
supervoxel_cluster_time	84
supervoxels	85
supervoxels_flash3d	88
synthetic_time_patterns	90
tessellate	91
test_all_clustering_methods	91
test_clustering_method	92
transform_new_data_svd	94
validate_cluster4d	94
Index	96

Description

Clusters fMRI voxels into spatially-coherent groups based on temporal correlation and spatial proximity. Includes optional refinement for boundary corrections. The algorithm supports parallel processing via the future package for improved performance.

Usage

```
acsc(
  bvec,
  mask,
  block_size = 2,
  ann_k = 10,
  alpha = 0.5,
  correlation_metric = c("pearson", "spearman", "robust"),
  spatial_weighting = c("gaussian", "binary"),
  refine = TRUE,
  max_refine_iter = 5,
  K = NULL,
  knn_proj_dim = 0L,
  knn_proj_method = c("none", "dct", "rp"),
  knn_proj_seed = 1L
)
```

Arguments

<code>bvec</code>	A NeuroVec-like object containing 4D fMRI data.
<code>mask</code>	A NeuroVol-like object (logical or numeric mask).
<code>block_size</code>	Approximate side length of blocks (e.g., 2 or 3). Must be > 0 .
<code>ann_k</code>	Number of approximate (or exact) nearest neighbors per block. Must be ≥ 1 .
<code>alpha</code>	Feature-similarity weight in $[0, 1]$. Zero is spatial-only and one is feature-only; both endpoints are supported.
<code>correlation_metric</code>	Similarity definition: Pearson correlation, Spearman correlation of average ranks, or robust biweight midcorrelation with tuning constant 9.
<code>spatial_weighting</code>	Spatial adjacency weighting ("gaussian", "binary").
<code>refine</code>	Logical; whether to refine boundaries.
<code>max_refine_iter</code>	Maximum iterations for boundary refinement. Must be ≥ 0 .
<code>K</code>	(Optional) Desired number of clusters.
<code>knn_proj_dim</code>	Optional integer; if > 0 , project block summaries to this dimension before kNN graph construction (speeds up FNN: <code>:get.knn()</code> for long time-series).
<code>knn_proj_method</code>	Projection method for <code>knn_proj_dim</code> : one of "none", "dct", or "rp" (random projection).
<code>knn_proj_seed</code>	Integer seed used when <code>knn_proj_method="rp"</code> (RNG state is restored after graph construction).

Details

ACSC builds feature-neighbor edges from the declared correlation metric and unions them with six-neighbor block-grid edges. Edge correlations are mapped to non-negative Louvain similarities as $(\text{correlation} + 1) / 2$. At $\alpha = 0$, feature-neighbor selection is bypassed entirely, so the graph topology and weights depend only on spatial adjacency. Optional DCT or random projection changes feature-neighbor candidate selection, but edge weights are always recomputed using the declared metric. Louvain calls use a fixed internal seed and restore the caller's RNG state, so identical ACSC inputs produce the same partition independently of the surrounding random stream.

Boundary refinement uses the same metric representation as graph building. Pearson uses centered unit vectors, Spearman uses centered average ranks, and robust mode uses median-centered Tukey-bisquare residuals. The C++ path accelerates centroid dot products and recomputes the active boundary after every synchronous pass; failures fall back to the equivalent R path.

After refinement, labels are flood-filled into connected components. If K is supplied, the shared adjacency-preserving exact- K engine merges or splits those components and fails closed when the target is topologically infeasible.

Value

A standardized `cluster4d_result` (also classed `acsc_result`) with final contiguous labels, original-space centers, physical coordinate centers, provenance, and these ACSC-specific elements:

cluster_map 3D array with cluster labels per voxel.

graph An `igraph` object used for clustering.

init_block_label Initial coarse partition (3D array) matching mask dimensions.

check_cluster_ordering

Check for Spatial Ordering in Cluster IDs

Description

Diagnostic function to detect whether cluster IDs are spatially ordered along any axis. Useful for identifying clustering algorithms that may produce ordered IDs.

Usage

```
check_cluster_ordering(cluster_result)
```

Arguments

cluster_result A clustering result object with `coord_centers` component

Value

A list with correlation statistics:

- cor_x: Spearman correlation between cluster ID and x-coordinate
- cor_y: Spearman correlation between cluster ID and y-coordinate
- cor_z: Spearman correlation between cluster ID and z-coordinate
- max_abs_cor: Maximum absolute correlation across all axes
- is_ordered: Logical indicating if any axis shows strong ordering ($|r| > 0.3$)

Examples

```
## Not run:
result <- cluster4d(vec, mask, K = 100, method = "flash3d")

# Check for ordering
ordering <- check_cluster_ordering(result)
print(ordering)

# If strongly ordered, randomize
if (ordering$is_ordered) {
  result <- randomize_cluster_ids(result)
}

## End(Not run)
```

```
cl_class_ids.cluster_result
```

Extract Class IDs from Cluster Result

Description

This function extracts the cluster class identifiers from a cluster result object. It is a method for the [cl_class_ids](#) generic from the clue package.

Usage

```
## S3 method for class 'cluster_result'
cl_class_ids(x)
```

Arguments

x A cluster_result object containing clustering information.

Value

An integer vector of cluster assignments, one for each data point.

See Also

[cl_class_ids](#) for the generic function.

cluster_metrics	<i>Explicit cluster-quality estimands</i>
-----------------	---

Description

Computes partition, within-cluster temporal, and physical spatial estimands from final voxel labels. Temporal coherence is the mean Pearson correlation over all unordered within-cluster voxel pairs. Spatial dispersion is the root mean squared Euclidean distance from each voxel coordinate to its final cluster centroid; lower values are more compact.

Usage

```
cluster_metrics(result, feature_mat = NULL, coords = NULL, truth = NULL)
```

Arguments

result	A <code>cluster_result</code> with one final label per voxel.
feature_mat	Optional finite numeric matrix with voxels in rows and at least two time/features in columns. A transposed matrix is accepted when only its columns match the number of labels. Every row must be non-constant for Pearson correlation.
coords	Optional finite numeric matrix with voxels in rows and spatial dimensions in columns. When omitted for a canonical <code>cluster4d_result</code> , physical millimetre coordinates are reconstructed from its mask and affine provenance.
truth	Optional partition labels over exactly the same voxels.

Value

A named list. Always contains `n_clusters` and `size_summary`. With `truth`, it adds `adjusted Rand`, `variation of information in bits`, and `pairwise Dice`. With `feature_mat`, it adds `temporal_pairwise_correlation`. With coordinates, it adds `spatial_rms_distance_mm` (or `spatial_rms_distance` when explicit coordinates are not declared to be millimetres by a canonical result).

cluster4d

*Unified 4D Clustering for Neuroimaging Data***Description**

Performs spatially-constrained clustering on 4D neuroimaging data using various algorithms. This is the main entry point for all clustering methods in the neurocluster package.

Usage

```
cluster4d(
  vec,
  mask,
  n_clusters = 100,
  method = c("supervoxels", "snic", "slic", "corr_slic", "brs_slic", "slice_msf",
    "flash3d", "g3s", "rena", "rena_plus", "mcl", "acsc", "commute"),
  spatial_weight = 0.5,
  max_iterations = NULL,
  connectivity = NULL,
  parallel = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

vec	A NeuroVec instance supplying the 4D data (x, y, z, time) to cluster
mask	A NeuroVol mask defining the voxels to include in clustering. Values must be finite. Voxels are included exactly when the mask value is strictly greater than zero; zero and negative values are excluded.
n_clusters	Finite integer target number of clusters (default 100), from one through the number of included mask voxels. Note that some methods may produce slightly different numbers of clusters due to algorithmic constraints.
method	Clustering algorithm to use. One of the 13 values below; the order is not a performance ranking or recommendation. • "supervoxels": Iterative heat kernel-based clustering (default) • "snic": Simple Non-Iterative Clustering • "slic": SLIC superpixels extended to 4D • "corr_slic": Correlation-embedded SLIC (local 3D SLIC updates on a compact random-projection embedding of the voxel time series) • "brs_slic": Boundary-Refined Sketch SLIC (coarse sketch + boundary exact-correlation refinement) • "slice_msf": Experimental Slice-wise Minimum Spanning Forest, available for explicit evaluation but not recommended • "flash3d": Fast Low-rank Approximate Superclusters

	• "g3s": Gradient-Guided Geodesic Supervoxels • "rena": Recursive Nearest Agglomeration (balanced, topology-aware) • "rena_plus": ReNA++, edge-aware multi-level ReNA coarsening followed by Ward refinement to the exact target • "mcl": Sparse Markov Cluster Algorithm on a weighted voxel graph • "acsc": Adaptive Correlation Superclustering (graph-based with boundary refinement) • "commute": Commute-time spectral clustering on a physical k-nearest-neighbor graph
spatial_weight	Method-specific spatial control in $[0, 1]$. For methods that define a convex feature/spatial blend, zero disables the spatial term and one disables the feature term. Other methods use the value as a bounded mapping to a native scale; their endpoints do not have that blend meaning. Default 0.5. This parameter is inactive for "rena" and "rena_plus"; supplying it explicitly for either method is an error. Maps to method-specific parameters: • supervoxels: $\alpha = 1 - \text{spatial_weight}$ (0 = all spatial, 1 = all feature) • snic: $\text{compactness} = \text{spatial_weight} * 10$ (range 0-10) • slic: $\text{compactness} = \text{spatial_weight} * 20$ (typical range 1-20) • corr_slic/brs_slic: direct convex blend between correlation and scaled spatial distance • slice_msf: $\text{compactness} = \text{spatial_weight} * 10$, followed by FH component scale $2 / (\text{compactness} + 1)$. This changes the graph-segmentation scale; it is not a convex spatial/feature blend. • flash3d: $\lambda_s = \text{spatial_weight}$ (direct mapping) • mcl: direct blend between feature and spatial edge similarities
max_iterations	Positive finite integer iteration limit for iterative methods. It is inactive for "snic", "slice_msf", and "commute"; supplying it explicitly for those methods is an error.
connectivity	Integer neighborhood connectivity. Supported values are method-specific: supervoxels accepts 6, 18, 26, or 27; slic, corr_slic, brs_slic, and slice_msf accept 6 or 26; G3S, ReNA variants, and MCL accept 6, 18, or 26. It is inactive for snic, flash3d, acsc, and commute.
parallel	Enable parallel processing for supervoxels and slic. For all other methods, an explicit TRUE is rejected; omitted or FALSE selects the serial implementation. Default TRUE for supported methods.
verbose	Print progress messages. Default FALSE.
...	Additional method-specific parameters. See method documentation for details.

Value

A cluster4d_result object (also inherits from cluster_result) containing:

labels	Contiguous positive integer final labels in mask voxel order
cluster	Backward-compatible alias that is identical to labels
clusvol	A ClusteredNeuroVol with cluster assignments

centers	Actual final-label means in the original feature space, with shape actual K by timepoints
coord_centers	Actual final-label coordinate means in physical mm, with shape actual K by 3
actual_k	Actual number of clusters produced
n_clusters	Backward-compatible alias of actual_k
label_ids	Explicit mapping from center rows to contiguous label IDs
method	Clustering method used
parameters	List of all parameters used
provenance	Typed label, original-feature, physical-coordinate, geometry, and mask provenance
metadata	Method-specific additional information

Algorithm Comparison

The *Speed* column reports single-threaded wall-clock times measured with `cluster4d()` defaults on the neuroim2 bundled mask (`global_mask_v4.nii`, 29,532 voxels), 20 timepoints of Gaussian noise and `n_clusters = 200`: *Fast* is under 1 s, *Moderate* a few seconds, and *Slow* tens of seconds. Timings depend on hardware, mask size, and parameters; treat the tiers as relative guidance, not a ranking of quality. *Parallel*, *Iterations*, and *Spatial weight* state whether the common `parallel`, `max_iterations`, and `spatial_weight` arguments are active.

Method	Speed	Parallel	Iterations	Spatial weight	Notes
supervoxels	Moderate	Yes	Yes	Yes	Heat-kernel iterative clustering; default method
snic	Slow	No	No	Yes	Non-iterative; run time is dominated by k-NN gradient search
slic	Fast	Yes	Yes	Yes	Compact 4D SLIC supervoxels
corr_slic	Fast	No	Yes	Yes	Correlation feature distance on a random-projection embedding
brs_slic	Fast	No	Yes	Yes	Sketch SLIC with exact-correlation boundary refinement
slice_msf	Fast	No	No	Yes	Experimental; explicit evaluation only
flash3d	Fast	No	Yes	Yes	Hash-based low-rank superclusters
g3s	Fast	No	Yes	Yes	Gradient-guided geodesic supervoxels
rena	Fast	No	Yes	No	Balanced, topology-aware agglomeration
rena_plus	Fast	No	Yes	No	Edge-aware ReNA coarsening with exact Ward refinement
mcl	Slow	No	Yes	Yes	Picks the natural MCL partition closest to <code>n_clusters</code> , <code>max_iterations</code>
acsc	Slow	No	Yes	Yes	Correlation graph with boundary refinement
commute	Moderate	No	No	Yes	Commuter-time spectral embedding; sparse eigensolver

Parameter Guidelines

For whole-brain parcellation:

- `n_clusters`: 100-1000 depending on desired granularity
- `spatial_weight`: 0.4-0.6 for balanced clustering
- `connectivity`: 26 for smoother boundaries

For ROI analysis:

- `n_clusters`: 10-100 depending on ROI size
- `spatial_weight`: 0.2-0.4 to emphasize functional similarity
- `connectivity`: 6 for more discrete parcels

For high-resolution data (< 2mm):

- `method`: "flash3d" for speed; Slice-MSF remains experimental
- `n_clusters`: Scale with voxel count (roughly `n_voxels/200`)

See Also

Method-specific functions: [cluster4d_supervoxels](#), [cluster4d_snic](#), [cluster4d_slic](#), [cluster4d_slice_msf](#), [cluster4d_flash3d](#), [cluster4d_g3s](#), [cluster4d_rena](#), [cluster4d_rena_plus](#), [cluster4d_mcl](#), [cluster4d_acsc](#), [cluster4d_commute](#)

Legacy functions (deprecated): [supervoxels](#), [snic](#), [slic4d_supervoxels](#), [slice_msf](#), [supervoxels_flash3d](#)

Examples

```
# Simple synthetic example (runs quickly)
library(neuroim2)
set.seed(1)
mask <- NeuroVol(array(1, c(4,4,4)), NeuroSpace(c(4,4,4)))
vec <- NeuroVec(array(rnorm(4*4*4*20), c(4,4,4,20)),
                  NeuroSpace(c(4,4,4,20)))
result <- cluster4d(vec, mask, n_clusters = 3, method = "g3s",
                   max_iterations = 1)
print(result$n_clusters)

# More realistic examples with larger data
mask <- NeuroVol(array(1, c(20,20,20)), NeuroSpace(c(20,20,20)))
vec <- replicate(50, NeuroVol(array(runif(20*20*20), c(20,20,20)),
                                NeuroSpace(c(20,20,20))), simplify=FALSE)
vec <- do.call(concat, vec)

# Basic usage with default supervoxels method
result <- cluster4d(vec, mask, n_clusters = 100)

# Fast clustering with FLASH-3D (hash-based)
result <- cluster4d(vec, mask, n_clusters = 100, method = "flash3d")

# Emphasize spatial compactness
result <- cluster4d(vec, mask, n_clusters = 100, spatial_weight = 0.8)

# Explicitly evaluate the experimental Slice-MSF method
result <- cluster4d(vec, mask, n_clusters = 100,
                   method = "slice_msf",
                   num_runs = 3, # slice_msf-specific parameter
                   consensus = TRUE)

# Get parameter suggestions for your data
```

```

n_vox <- sum(mask > 0)
n_time <- dim(vec)[4]
params <- suggest_cluster4d_params(n_vox, n_time, priority = "quality")
result <- cluster4d(vec, mask,
                    n_clusters = params$n_clusters,
                    method = params$recommended_method)

```

cluster4d_acsc

*Cluster4d using ACSC method***Description**

Wrapper for Adaptive Correlation Superclustering algorithm with standardized interface. ACSC uses graph-based clustering with Louvain community detection and optional boundary refinement.

Usage

```

cluster4d_acsc(
  vec,
  mask,
  n_clusters = 100,
  spatial_weight = 0.5,
  max_iterations = 5,
  verbose = FALSE,
  block_size = 2,
  correlation_metric = c("pearson", "spearman", "robust"),
  spatial_weighting = c("gaussian", "binary"),
  refine = TRUE,
  max_refine_iter = NULL,
  ...
)

```

Arguments

<code>vec</code>	A NeuroVec instance supplying the 4D data (x, y, z, time) to cluster
<code>mask</code>	A NeuroVol mask defining the voxels to include in clustering. Values must be finite. Voxels are included exactly when the mask value is strictly greater than zero; zero and negative values are excluded.
<code>n_clusters</code>	Finite integer target number of clusters (default 100), from one through the number of included mask voxels. Note that some methods may produce slightly different numbers of clusters due to algorithmic constraints.
<code>spatial_weight</code>	Method-specific spatial control in $[0, 1]$. For methods that define a convex feature/spatial blend, zero disables the spatial term and one disables the feature term. Other methods use the value as a bounded mapping to a native scale; their endpoints do not have that blend meaning. Default 0.5. This parameter is inactive for "rena" and "rena_plus"; supplying it explicitly for either method is an error. Maps to method-specific parameters:

	• supervoxels: $\alpha = 1 - \text{spatial_weight}$ (0 = all spatial, 1 = all feature) • snic: $\text{compactness} = \text{spatial_weight} * 10$ (range 0-10) • slic: $\text{compactness} = \text{spatial_weight} * 20$ (typical range 1-20) • corr_slic/brs_slic: direct convex blend between correlation and scaled spatial distance • slice_msf: $\text{compactness} = \text{spatial_weight} * 10$, followed by FH component scale $2 / (\text{compactness} + 1)$. This changes the graph-segmentation scale; it is not a convex spatial/feature blend. • flash3d: $\lambda_s = \text{spatial_weight}$ (direct mapping) • mcl: direct blend between feature and spatial edge similarities
max_iterations	Positive finite integer iteration limit for iterative methods. It is inactive for "snic", "slice_msf", and "commute"; supplying it explicitly for those methods is an error.
verbose	Print progress messages. Default FALSE.
block_size	Approximate side length of initial blocks. Default 2.
correlation_metric	Correlation definition used consistently for graph construction and refinement: "pearson", "spearman", or "robust".
spatial_weighting	Spatial block-edge weights: "gaussian" or "binary".
refine	Logical; whether to refine boundaries. Default TRUE.
max_refine_iter	Maximum iterations for boundary refinement. Default 5.
...	Additional parameters passed to acsc

Value

A cluster4d_result object

cluster4d_brsslic	<i>Cluster4d using boundary-refined sketch SLIC</i>
-------------------	---

Description

Hybrid method: fast sketch-SLIC coarse assignment, then exact-correlation refinement only on boundary voxels.

Usage

```
cluster4d_brsslic(
  vec,
  mask,
  n_clusters = 100,
  spatial_weight = 0.05,
```

```

    max_iterations = 2,
    connectivity = 6,
    parallel = FALSE,
    verbose = FALSE,
    embedding_dim = 24L,
    sketch_repeats = 1L,
    coarse_alpha = NULL,
    boundary_passes = 1L,
    global_passes = 0L,
    refine_spatial_weight = NULL,
    refine_l2_weight = 0,
    refine_stride = NULL,
    seed = 1L,
    min_size = NULL,
    ...
)

```

Arguments

<code>vec</code>	A <code>NeuroVec</code> instance supplying the 4D data (x, y, z, time) to cluster
<code>mask</code>	A <code>NeuroVol</code> mask defining the voxels to include in clustering. Values must be finite. Voxels are included exactly when the mask value is strictly greater than zero; zero and negative values are excluded.
<code>n_clusters</code>	Finite integer target number of clusters (default 100), from one through the number of included mask voxels. Note that some methods may produce slightly different numbers of clusters due to algorithmic constraints.
<code>spatial_weight</code>	Method-specific spatial control in $[0, 1]$. For methods that define a convex feature/spatial blend, zero disables the spatial term and one disables the feature term. Other methods use the value as a bounded mapping to a native scale; their endpoints do not have that blend meaning. Default 0.5. This parameter is inactive for "rena" and "rena_plus"; supplying it explicitly for either method is an error. Maps to method-specific parameters: • <code>supervoxels</code>: $\alpha = 1 - \text{spatial_weight}$ (0 = all spatial, 1 = all feature) • <code>snic</code>: $\text{compactness} = \text{spatial_weight} * 10$ (range 0-10) • <code>slic</code>: $\text{compactness} = \text{spatial_weight} * 20$ (typical range 1-20) • <code>corr_slic/brs_slic</code>: direct convex blend between correlation and scaled spatial distance • <code>slice_msf</code>: $\text{compactness} = \text{spatial_weight} * 10$, followed by FH component scale $2 / (\text{compactness} + 1)$. This changes the graph-segmentation scale; it is not a convex spatial/feature blend. • <code>flash3d</code>: $\lambda_s = \text{spatial_weight}$ (direct mapping) • <code>mcl</code>: direct blend between feature and spatial edge similarities
<code>max_iterations</code>	Positive finite integer iteration limit for iterative methods. It is inactive for "snic", "slice_msf", and "commute"; supplying it explicitly for those methods is an error.

connectivity	Integer neighborhood connectivity. Supported values are method-specific: supervoxels accepts 6, 18, 26, or 27; slic, corr_slic, brs_slic, and slice_msf accept 6 or 26; G3S, ReNA variants, and MCL accept 6, 18, or 26. It is inactive for snic, flash3d, acsc, and commute.
parallel	Enable parallel processing for supervoxels and slic. For all other methods, an explicit TRUE is rejected; omitted or FALSE selects the serial implementation. Default TRUE for supported methods.
verbose	Print progress messages. Default FALSE.
embedding_dim	Embedding dimension for coarse sketch stage.
sketch_repeats	Number of independent sketches to average.
coarse_alpha	Optional backward-compatible alias for the coarse spatial blend weight in $[0, 1]$.
boundary_passes	Number of boundary-only refinement passes.
global_passes	Number of local-global refinement passes after boundary passes.
refine_spatial_weight	Optional spatial blend weight during exact refinement. NULL inherits coarse_alpha; both endpoints are supported.
refine_l2_weight	Optional L2 prototype penalty during boundary refinement.
refine_stride	Optional temporal stride for refinement (NULL = auto).
seed	Random seed.
min_size	Minimum component size for connectivity enforcement.
...	Reserved for future extensions.

Details

Both the coarse and exact-refinement spatial weights use the convex blend $(1 - w) * \text{correlation_distance} + w * \text{scaled_spatial_distance}$. `refine_l2_weight` is a separate optional feature penalty, so a strictly spatial-only refinement uses `refine_spatial_weight = 1` and `refine_l2_weight = 0`. The hash-sketch artifact is explicitly marked as a pre-refinement embedding under `metadata$approximation$coarse_embe`. Public centers and coordinates are final-label means in original feature and physical coordinate spaces, respectively.

Value

A `cluster4d_result` object

cluster4d_commute	<i>Cluster4d using commute-time spectral method</i>
-------------------	---

Description

Wrapper for commute_cluster with standardized interface.

Usage

```
cluster4d_commute(
    vec,
    mask,
    n_clusters = 100,
    spatial_weight = 0.5,
    verbose = FALSE,
    ncomp = NULL,
    ...
)
```

Arguments

vec	A NeuroVec instance supplying the 4D data (x, y, z, time) to cluster
mask	A NeuroVol mask defining the voxels to include in clustering. Values must be finite. Voxels are included exactly when the mask value is strictly greater than zero; zero and negative values are excluded.
n_clusters	Finite integer target number of clusters (default 100), from one through the number of included mask voxels. Note that some methods may produce slightly different numbers of clusters due to algorithmic constraints.
spatial_weight	Method-specific spatial control in $[0, 1]$. For methods that define a convex feature/spatial blend, zero disables the spatial term and one disables the feature term. Other methods use the value as a bounded mapping to a native scale; their endpoints do not have that blend meaning. Default 0.5. This parameter is inactive for "rena" and "rena_plus"; supplying it explicitly for either method is an error. Maps to method-specific parameters: • supervoxels: $\alpha = 1 - \text{spatial_weight}$ (0 = all spatial, 1 = all feature) • snic: $\text{compactness} = \text{spatial_weight} * 10$ (range 0-10) • slic: $\text{compactness} = \text{spatial_weight} * 20$ (typical range 1-20) • corr_slic/brs_slic: direct convex blend between correlation and scaled spatial distance • slice_msf: $\text{compactness} = \text{spatial_weight} * 10$, followed by FH component scale $2 / (\text{compactness} + 1)$. This changes the graph-segmentation scale; it is not a convex spatial/feature blend. • flash3d: $\lambda_s = \text{spatial_weight}$ (direct mapping) • mcl: direct blend between feature and spatial edge similarities
verbose	Print progress messages. Default FALSE.

ncomp Number of embedding components (defaults to $\sqrt{2K}$).
 ... Additional method-specific parameters. See method documentation for details.

Value

A cluster4d_result object

Note

Masks with more than 3,000 voxels use a sparse shift-invert eigensolver (the required package **eigencore**); smaller masks use the dense `eigen()` path. Pass `eigen_solver` (see [commute_cluster](#)) to choose explicitly.

cluster4d_corrslc	<i>Cluster4d using correlation-embedded SLIC method</i>
-------------------	---

Description

Wrapper for a correlation-first SLIC variant that builds a compact random projection embedding of voxel time series, then runs local 3D SLIC updates.

Usage

```
cluster4d_corrslc(
  vec,
  mask,
  n_clusters = 100,
  spatial_weight = 0.5,
  max_iterations = 6,
  connectivity = 6,
  parallel = FALSE,
  verbose = FALSE,
  embedding_dim = 64L,
  adaptive_embedding = FALSE,
  embedding_basis = c("hash", "dct"),
  embedding_whiten = FALSE,
  sketch_repeats = 1L,
  alpha = NULL,
  assign_stride = 1L,
  quantize_assign = FALSE,
  refine_exact_iters = 0L,
  refine_boundary_only = TRUE,
  refine_stride = 1L,
  refine_alpha = NULL,
  seed = 1L,
  min_size = NULL,
  ...,
  .input_contract = NULL
)
```

Arguments

<code>vec</code>	A NeuroVec instance supplying the 4D data (x, y, z, time) to cluster
<code>mask</code>	A NeuroVol mask defining the voxels to include in clustering. Values must be finite. Voxels are included exactly when the mask value is strictly greater than zero; zero and negative values are excluded.
<code>n_clusters</code>	Finite integer target number of clusters (default 100), from one through the number of included mask voxels. Note that some methods may produce slightly different numbers of clusters due to algorithmic constraints.
<code>spatial_weight</code>	Method-specific spatial control in $[0, 1]$. For methods that define a convex feature/spatial blend, zero disables the spatial term and one disables the feature term. Other methods use the value as a bounded mapping to a native scale; their endpoints do not have that blend meaning. Default 0.5. This parameter is inactive for "rena" and "rena_plus"; supplying it explicitly for either method is an error. Maps to method-specific parameters: • supervoxels: $\alpha = 1 - \text{spatial_weight}$ (0 = all spatial, 1 = all feature) • snic: $\text{compactness} = \text{spatial_weight} * 10$ (range 0-10) • slic: $\text{compactness} = \text{spatial_weight} * 20$ (typical range 1-20) • corr_slic/brs_slic: direct convex blend between correlation and scaled spatial distance • slice_msf: $\text{compactness} = \text{spatial_weight} * 10$, followed by FH component scale $2 / (\text{compactness} + 1)$. This changes the graph-segmentation scale; it is not a convex spatial/feature blend. • flash3d: $\lambda_s = \text{spatial_weight}$ (direct mapping) • mcl: direct blend between feature and spatial edge similarities
<code>max_iterations</code>	Positive finite integer iteration limit for iterative methods. It is inactive for "snic", "slice_msf", and "commute"; supplying it explicitly for those methods is an error.
<code>connectivity</code>	Integer neighborhood connectivity. Supported values are method-specific: supervoxels accepts 6, 18, 26, or 27; slic, corr_slic, brs_slic, and slice_msf accept 6 or 26; G3S, ReNA variants, and MCL accept 6, 18, or 26. It is inactive for snic, flash3d, acsc, and commute.
<code>parallel</code>	Enable parallel processing for supervoxels and slic. For all other methods, an explicit TRUE is rejected; omitted or FALSE selects the serial implementation. Default TRUE for supported methods.
<code>verbose</code>	Print progress messages. Default FALSE.
<code>embedding_dim</code>	Embedding dimension used to approximate correlations. Use "auto" (or set <code>adaptive_embedding = TRUE</code>) for data-adaptive selection.
<code>adaptive_embedding</code>	Logical; if TRUE, choose embedding dimension from data size and time length.
<code>embedding_basis</code>	Embedding basis, either "hash" (CountSketch-like) or "dct" (demeaned DCT basis).
<code>embedding_whiten</code>	Logical; if TRUE, whiten embedding dimensions across voxels.

sketch_repeats	Number of independent hash sketches combined per voxel.
alpha	Optional backward-compatible alias for the spatial blend weight. It must be in $[0, 1]$: 0 is correlation-only and 1 is spatial-only.
assign_stride	Assignment subsampling stride for coarse updates. 1 disables subsampling; values > 1 process one z-slice phase per iteration followed by a final full assignment pass.
quantize_assign	Logical; if TRUE, quantize embeddings and centroid features to int8 during assignment for faster dot-product distance checks.
refine_exact_iters	Number of exact-correlation refinement passes after coarse embedding-SLIC iterations.
refine_boundary_only	Logical; if TRUE, exact refinement is restricted to boundary voxels each pass.
refine_stride	Time-axis subsampling stride for exact refinement. Values > 1 speed up refinement by computing correlations on every refine_stride-th time point (approximate).
refine_alpha	Optional spatial blend weight for exact refinement. If NULL, uses alpha; both endpoints are supported.
seed	Random seed for embedding hash and seed initialization.
min_size	Minimum component size for connectivity enforcement.
...	Additional arguments (currently unused; reserved for forward compatibility).
.input_contract	Internal prevalidated input receipt used by cluster4d(); direct callers should leave this as NULL.

Details

spatial_weight (or its alias alpha) is a convex blend: $(1 - w) * \text{correlation_distance} + w * \text{scaled_spatial_distance}$. Coarse correlation is approximated by a hash or DCT sketch. Assignment may additionally use int8 quantization or z-plane subsampling; exact refinement may use a temporal stride. These choices and their stage are recorded under `metadata$approximation`. The public centers are always recomputed from the final labels in the original time-series space; sketch centers are separately typed metadata and are never substituted for them.

Value

A cluster4d_result object

cluster4d_flash3d

*Cluster4d using FLASH-3D method***Description**

Wrapper for FLASH-3D algorithm with standardized interface.

Usage

```
cluster4d_flash3d(
    vec,
    mask,
    n_clusters = 100,
    spatial_weight = 0.25,
    max_iterations = 4,
    verbose = FALSE,
    lambda_t = NULL,
    bits = 64,
    dctM = 16,
    ...
)
```

Arguments

vec	A NeuroVec instance supplying the 4D data (x, y, z, time) to cluster
mask	A NeuroVol mask defining the voxels to include in clustering. Values must be finite. Voxels are included exactly when the mask value is strictly greater than zero; zero and negative values are excluded.
n_clusters	Finite integer target number of clusters (default 100), from one through the number of included mask voxels. Note that some methods may produce slightly different numbers of clusters due to algorithmic constraints.
spatial_weight	Method-specific spatial control in $[0, 1]$. For methods that define a convex feature/spatial blend, zero disables the spatial term and one disables the feature term. Other methods use the value as a bounded mapping to a native scale; their endpoints do not have that blend meaning. Default 0.5. This parameter is inactive for "rena" and "rena_plus"; supplying it explicitly for either method is an error. Maps to method-specific parameters: - supervoxels: $\alpha = 1 - \text{spatial_weight}$ (0 = all spatial, 1 = all feature) - snic: $\text{compactness} = \text{spatial_weight} * 10$ (range 0-10) - slic: $\text{compactness} = \text{spatial_weight} * 20$ (typical range 1-20) - corr_slic/brs_slic: direct convex blend between correlation and scaled spatial distance - slice_msf: $\text{compactness} = \text{spatial_weight} * 10$, followed by FH component scale $2 / (\text{compactness} + 1)$. This changes the graph-segmentation scale; it is not a convex spatial/feature blend.

	• flash3d: <code>lambda_s = spatial_weight</code> (direct mapping) • mcl: direct blend between feature and spatial edge similarities
<code>max_iterations</code>	Positive finite integer iteration limit for iterative methods. It is inactive for "snic", "slice_msf", and "commute"; supplying it explicitly for those methods is an error.
<code>verbose</code>	Print progress messages. Default FALSE.
<code>lambda_t</code>	Optional finite non-negative temporal weight for Hamming distance. When omitted, it is $1 - \text{spatial_weight}$, so the unified interface preserves both feature-only and spatial-only endpoints.
<code>bits</code>	Hash length (64 or 128)
<code>dctM</code>	Number of DCT coefficients
<code>...</code>	Additional parameters passed to <code>supervoxels_flash3d</code>

Value

A `cluster4d_result` object

<code>cluster4d_g3s</code>	<i>Gradient-guided geodesic clustering on a masked grid</i>
----------------------------	---

Description

Compresses voxel features, selects low-gradient seeds, and assigns voxels by deterministic multi-source Dijkstra propagation. Edges are exactly the requested masked-grid neighbors; excluded gaps are never bridged. The edge objective is $\alpha * \text{feature_distance} + (1 - \alpha) * \text{physical_distance} / \text{compactness}$.

Usage

```
cluster4d_g3s(
  vec,
  mask,
  K = 100,
  n_components = 15,
  variance_threshold = 0.95,
  alpha = 0.5,
  compactness = NULL,
  max_refinement_iter = 3,
  verbose = FALSE,
  use_irlba = TRUE,
  use_rsvd = TRUE,
  connectivity = 26
)
```

Arguments

vec	A NeuroVec or SparseNeuroVec containing the input features.
mask	A NeuroVol; exactly finite values greater than zero are included.
K	Positive requested cluster count.
n_components	Requested SVD feature count.
variance_threshold	Minimum retained variance in $[0, 1]$.
alpha	Feature-distance weight in $[0, 1]$; zero is spatial-only and one is feature-only.
compactness	Positive finite physical length scale. Smaller values impose a stronger spatial penalty. The default uses masked voxel measure, K, and the effective dimension, so single-slice masks remain well-defined.
max_refinement_iter	Non-negative number of boundary passes. Refinement uses the same feature/spatial mixture and final labels are repaired through the shared adjacency-preserving exact-K engine.
verbose	Whether to report phase progress.
use_irlba	Whether large SVDs may use irlba.
use_rsvd	Whether randomized SVD may be used when available.
connectivity	Exact grid connectivity: 6, 18, or 26.

Value

A typed `g3s_result` and `cluster4d_result` with K by T centers, physical coordinate centers, graph provenance, and physical-scale metadata.

References

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 269-271.

cluster4d_mcl	<i>Cluster4d using sparse Markov Clustering (MCL)</i>
---------------	---

Description

Builds a sparse voxel graph, runs an MCL flow process, and maps attractors to cluster labels. The implementation is optimized for sparse neuroimaging graphs.

Usage

```
cluster4d_mcl(
  vec,
  mask,
  n_clusters = 100,
  spatial_weight = 0.2,
  max_iterations = 8,
  connectivity = 6,
  verbose = FALSE,
  inflation = 1.6,
  expansion = 2L,
  loop_weight = 1,
  prune_k = NULL,
  prune_threshold = 1e-06,
  tol = 1e-04,
  feature_metric = c("correlation", "euclidean"),
  feature_sigma = NULL,
  spatial_sigma = NULL,
  exact_k = FALSE
)
```

Arguments

<code>vec</code>	A <code>NeuroVec</code> , <code>SparseNeuroVec</code> , or <code>NeuroVol</code> containing the features.
<code>mask</code>	A <code>NeuroVol</code> mask; finite values strictly greater than zero are included.
<code>n_clusters</code>	Requested cluster count. With <code>exact_k = FALSE</code> , this is the target used to select the closest natural MCL partition across a fixed, deterministic inflation search. With <code>exact_k = TRUE</code> , it is an exact adjacency-preserving postcondition.
<code>spatial_weight</code>	Convex weight from zero through one for physical edge similarity.
<code>max_iterations</code>	Positive MCL iteration limit for each inflation candidate.
<code>connectivity</code>	Exact masked grid connectivity: 6, 18, or 26.
<code>verbose</code>	Logical; report iteration diagnostics.
<code>inflation</code>	Nominal MCL inflation parameter (>1). In natural target mode, the deterministic search evaluates this value and fixed multipliers around it.
<code>expansion</code>	MCL expansion power (integer ≥ 2). Default 2.
<code>loop_weight</code>	Finite non-negative self-loop weight added before normalization.
<code>prune_k</code>	Maximum nonzero entries kept per column during MCL iterations. Must be a positive integer; the default is 64 and is independent of the target K .
<code>prune_threshold</code>	Finite threshold in $[0, 1)$. The strongest entry in every column is retained even when all entries fall below the threshold.
<code>tol</code>	Positive convergence tolerance on maximum absolute matrix delta.
<code>feature_metric</code>	Feature similarity metric: "correlation" or "euclidean".
<code>feature_sigma</code>	Optional bandwidth for euclidean heat-kernel feature similarity.
<code>spatial_sigma</code>	Optional bandwidth for spatial heat-kernel similarity.
<code>exact_k</code>	If <code>TRUE</code> , use the shared adjacency-preserving exact- K engine.

Details

Every expansion, inflation, and pruning step is checked to remain finite, non-negative, and column stochastic. The `parallel` argument was removed because this sparse Matrix implementation has no parallel backend. Supplying it is therefore an ordinary unused-argument error rather than a silently sequential execution.

Value

A `cluster4d_result`. Metadata records natural and final K, convergence, selected inflation, candidate search results, flow sparsity, final normalization error, and exact-K policy.

cluster4d_rena	<i>Cluster4d using ReNA method</i>
----------------	------------------------------------

Description

Wrapper for ReNA algorithm with standardized cluster4d interface.

Usage

```
cluster4d_rena(
  vec,
  mask,
  n_clusters = 100,
  max_iterations = 50,
  connectivity = 26,
  verbose = FALSE,
  exact_k = TRUE,
  ...
)
```

Arguments

<code>vec</code>	A <code>NeuroVec</code> instance supplying the 4D data (x, y, z, time) to cluster
<code>mask</code>	A <code>NeuroVol</code> mask defining the voxels to include in clustering. Values must be finite. Voxels are included exactly when the mask value is strictly greater than zero; zero and negative values are excluded.
<code>n_clusters</code>	Finite integer target number of clusters (default 100), from one through the number of included mask voxels. Note that some methods may produce slightly different numbers of clusters due to algorithmic constraints.
<code>max_iterations</code>	Positive finite integer iteration limit for iterative methods. It is inactive for "snic", "slice_msf", and "commute"; supplying it explicitly for those methods is an error.
<code>connectivity</code>	Integer neighborhood connectivity. Supported values are method-specific: supervoxels accepts 6, 18, 26, or 27; slic, corr_slic, brs_slic, and slice_msf accept 6 or 26; G3S, ReNA variants, and MCL accept 6, 18, or 26. It is inactive for snic, flash3d, acsc, and commute.

verbose	Print progress messages. Default FALSE.
exact_k	Logical; if TRUE, apply edge pruning to target exactly n_clusters.
...	Additional parameters (currently unused for ReNA). ReNA has no spatial/feature trade-off, so spatial_weight is not a formal argument; supplying it through ... is an error.

Value

A cluster4d_result object

cluster4d_rena_plus	<i>Cluster4d wrapper for ReNA++</i>
---------------------	-------------------------------------

Description

Cluster4d wrapper for ReNA++

Usage

```
cluster4d_rena_plus(
  vec,
  mask,
  n_clusters = 100,
  r = 2,
  lambda = 1,
  grad_img = NULL,
  connectivity = 26,
  max_iterations = 50,
  verbose = FALSE,
  ...
)
```

Arguments

vec	A NeuroVec instance supplying the 4D data (x, y, z, time) to cluster
mask	A NeuroVol mask defining the voxels to include in clustering. Values must be finite. Voxels are included exactly when the mask value is strictly greater than zero; zero and negative values are excluded.
n_clusters	Exact finite integer target number of clusters, from one through the number of included mask voxels.
r	Over-clustering factor for coarsening stage.
lambda	Gradient penalty weight.
grad_img	Optional gradient/intensity vector (see rena_plus).

connectivity	Integer neighborhood connectivity. Supported values are method-specific: supervoxels accepts 6, 18, 26, or 27; slic, corr_slic, brs_slic, and slice_msf accept 6 or 26; G3S, ReNA variants, and MCL accept 6, 18, or 26. It is inactive for snic, flash3d, acsc, and commute.
max_iterations	Positive finite integer iteration limit for iterative methods. It is inactive for "snic", "slice_msf", and "commute"; supplying it explicitly for those methods is an error.
verbose	Print progress messages. Default FALSE.
...	Additional arguments passed to rena_plus(). ReNA++ has no spatial/feature trade-off, so spatial_weight is not a formal argument; supplying it through ... is an error.

Value

A cluster4d_result.

cluster4d_slic	<i>Cluster4d using SLIC method</i>
----------------	------------------------------------

Description

Wrapper for SLIC algorithm with standardized interface.

Usage

```
cluster4d_slic(
  vec,
  mask,
  n_clusters = 100,
  spatial_weight = 0.5,
  max_iterations = 10,
  connectivity = 26,
  parallel = TRUE,
  verbose = FALSE,
  preserve_k = FALSE,
  seed_relocate = "none",
  ...,
  .input_contract = NULL
)
```

Arguments

vec	A NeuroVec instance supplying the 4D data (x, y, z, time) to cluster
mask	A NeuroVol mask defining the voxels to include in clustering. Values must be finite. Voxels are included exactly when the mask value is strictly greater than zero; zero and negative values are excluded.

n_clusters	Finite integer target number of clusters (default 100), from one through the number of included mask voxels. Note that some methods may produce slightly different numbers of clusters due to algorithmic constraints.
spatial_weight	Method-specific spatial control in $[0, 1]$. For methods that define a convex feature/spatial blend, zero disables the spatial term and one disables the feature term. Other methods use the value as a bounded mapping to a native scale; their endpoints do not have that blend meaning. Default 0.5. This parameter is inactive for "rena" and "rena_plus"; supplying it explicitly for either method is an error. Maps to method-specific parameters: • supervoxels: $\alpha = 1 - \text{spatial_weight}$ (0 = all spatial, 1 = all feature) • snic: $\text{compactness} = \text{spatial_weight} * 10$ (range 0-10) • slic: $\text{compactness} = \text{spatial_weight} * 20$ (typical range 1-20) • corr_slic/brs_slic: direct convex blend between correlation and scaled spatial distance • slice_msf: $\text{compactness} = \text{spatial_weight} * 10$, followed by FH component scale $2 / (\text{compactness} + 1)$. This changes the graph-segmentation scale; it is not a convex spatial/feature blend. • flash3d: $\lambda_s = \text{spatial_weight}$ (direct mapping) • mcl: direct blend between feature and spatial edge similarities
max_iterations	Positive finite integer iteration limit for iterative methods. It is inactive for "snic", "slice_msf", and "commute"; supplying it explicitly for those methods is an error.
connectivity	Integer neighborhood connectivity. Supported values are method-specific: supervoxels accepts 6, 18, 26, or 27; slic, corr_slic, brs_slic, and slice_msf accept 6 or 26; G3S, ReNA variants, and MCL accept 6, 18, or 26. It is inactive for snic, flash3d, acsc, and commute.
parallel	Enable parallel processing for supervoxels and slic. For all other methods, an explicit TRUE is rejected; omitted or FALSE selects the serial implementation. Default TRUE for supported methods.
verbose	Print progress messages. Default FALSE.
preserve_k	Request exactly K connected clusters. Exact K is feasible only when K is at least the number of connected mask components; below that topological minimum, connectivity takes precedence and the minimum feasible cluster count is returned with a warning.
seed_relocate	Seed relocation method
...	Additional parameters passed to slic4d_supervoxels
.input_contract	Internal prevalidated input receipt used by cluster4d(); direct callers should leave this as NULL.

Value

A cluster4d_result object

cluster4d_slice_msf *Cluster4d using slice_msf method*

Description

Wrapper for slice_msf algorithm with standardized interface. Slice-MSF is experimental and is deliberately excluded from [suggest_cluster4d_params\(\)](#) recommendations pending release re-certification.

Usage

```
cluster4d_slice_msf(
    vec,
    mask,
    n_clusters = 100,
    spatial_weight = 0.2,
    connectivity = 26,
    parallel = TRUE,
    verbose = FALSE,
    num_runs = 1,
    consensus = FALSE,
    stitch_z = TRUE,
    r = 12,
    gamma = 0,
    z_mult = 0,
    min_size = NULL,
    seed = 1L,
    ensemble_fraction = 0.8,
    ...
)
```

Arguments

vec	A NeuroVec instance supplying the 4D data (x, y, z, time) to cluster
mask	A NeuroVol mask defining the voxels to include in clustering. Values must be finite. Voxels are included exactly when the mask value is strictly greater than zero; zero and negative values are excluded.
n_clusters	Finite integer target number of clusters (default 100), from one through the number of included mask voxels. Note that some methods may produce slightly different numbers of clusters due to algorithmic constraints.
spatial_weight	Method-specific spatial control in $[0, 1]$. For methods that define a convex feature/spatial blend, zero disables the spatial term and one disables the feature term. Other methods use the value as a bounded mapping to a native scale; their endpoints do not have that blend meaning. Default 0.5. This parameter is inactive for "rena" and "rena_plus"; supplying it explicitly for either method is an error. Maps to method-specific parameters:

	• supervoxels: $\alpha = 1 - \text{spatial_weight}$ (0 = all spatial, 1 = all feature) • snic: $\text{compactness} = \text{spatial_weight} * 10$ (range 0-10) • slic: $\text{compactness} = \text{spatial_weight} * 20$ (typical range 1-20) • corr_slic/brs_slic: direct convex blend between correlation and scaled spatial distance • slice_msf: $\text{compactness} = \text{spatial_weight} * 10$, followed by FH component scale $2 / (\text{compactness} + 1)$. This changes the graph-segmentation scale; it is not a convex spatial/feature blend. • flash3d: $\lambda_s = \text{spatial_weight}$ (direct mapping) • mcl: direct blend between feature and spatial edge similarities
connectivity	Integer neighborhood connectivity. Supported values are method-specific: supervoxels accepts 6, 18, 26, or 27; slic, corr_slic, brs_slic, and slice_msf accept 6 or 26; G3S, ReNA variants, and MCL accept 6, 18, or 26. It is inactive for snic, flash3d, acsc, and commute.
parallel	Enable parallel processing for supervoxels and slic. For all other methods, an explicit TRUE is rejected; omitted or FALSE selects the serial implementation. Default TRUE for supported methods.
verbose	Print progress messages. Default FALSE.
num_runs	Number of independent runs
consensus	Use consensus fusion
stitch_z	Whether graph edges may cross axial slices.
r	Number of non-DC DCT modes.
gamma	Reserved compatibility parameter; must be zero.
z_mult	Axial DCT-sketch smoothing fraction in $[0, 1]$; zero leaves sketches unsmoothed.
min_size	Minimum component size after connectivity enforcement. Default NULL (auto).
seed	Integer seed for multi-run DCT subspaces.
ensemble_fraction	Candidate-pool fraction for multi-run DCT subspaces.
...	Additional parameters passed to slice_msf

Details

Unlike direct `slice_msf()`, this wrapper always requests exactly `n_clusters`, defaults to one non-consensus run, and chooses a conservative automatic `min_size`. `spatial_weight` maps to $\text{compactness} = \text{spatial_weight} * 10$ and then to FH scale $2 / (\text{compactness} + 1)$; it is not a convex spatial/feature blend. Positive `gamma` values fail with class `slice_msf_unsupported_gamma`.

Value

A `cluster4d_result` object with natural/exact-K and effective-control provenance in `metadata$exact_k_repair`; see `slice_msf()`.

cluster4d_snic	<i>Cluster4d using SNIC method</i>
----------------	------------------------------------

Description

Wrapper for SNIC algorithm with standardized interface.

Usage

```
cluster4d_snic(
    vec,
    mask,
    n_clusters = 100,
    spatial_weight = 0.5,
    max_iterations = 100,
    verbose = FALSE,
    ...
)
```

Arguments

vec	A NeuroVec instance supplying the 4D data (x, y, z, time) to cluster
mask	A NeuroVol mask defining the voxels to include in clustering. Values must be finite. Voxels are included exactly when the mask value is strictly greater than zero; zero and negative values are excluded.
n_clusters	Finite integer target number of clusters (default 100), from one through the number of included mask voxels. Note that some methods may produce slightly different numbers of clusters due to algorithmic constraints.
spatial_weight	Method-specific spatial control in $[0, 1]$. For methods that define a convex feature/spatial blend, zero disables the spatial term and one disables the feature term. Other methods use the value as a bounded mapping to a native scale; their endpoints do not have that blend meaning. Default 0.5. This parameter is inactive for "rena" and "rena_plus"; supplying it explicitly for either method is an error. Maps to method-specific parameters: • supervoxels: $\alpha = 1 - \text{spatial_weight}$ (0 = all spatial, 1 = all feature) • snic: $\text{compactness} = \text{spatial_weight} * 10$ (range 0-10) • slic: $\text{compactness} = \text{spatial_weight} * 20$ (typical range 1-20) • corr_slic/brs_slic: direct convex blend between correlation and scaled spatial distance • slice_msf: $\text{compactness} = \text{spatial_weight} * 10$, followed by FH component scale $2 / (\text{compactness} + 1)$. This changes the graph-segmentation scale; it is not a convex spatial/feature blend. • flash3d: $\lambda_s = \text{spatial_weight}$ (direct mapping) • mcl: direct blend between feature and spatial edge similarities

max_iterations	Positive finite integer iteration limit for iterative methods. It is inactive for "snic", "slice_msf", and "commute"; supplying it explicitly for those methods is an error.
verbose	Print progress messages. Default FALSE.
...	Additional parameters passed to snic

Value

A cluster4d_result object

cluster4d_supervoxels *Cluster4d using supervoxels method*

Description

Wrapper for supervoxels algorithm with standardized interface.

Usage

```
cluster4d_supervoxels(
  vec,
  mask,
  n_clusters = 100,
  spatial_weight = 0.5,
  max_iterations = 50,
  connectivity = 27,
  parallel = TRUE,
  verbose = FALSE,
  sigma1 = 1,
  sigma2 = 2.5,
  use_gradient = TRUE,
  converge_thresh = 0.001,
  ...
)
```

Arguments

vec	A NeuroVec instance supplying the 4D data (x, y, z, time) to cluster
mask	A NeuroVol mask defining the voxels to include in clustering. Values must be finite. Voxels are included exactly when the mask value is strictly greater than zero; zero and negative values are excluded.
n_clusters	Finite integer target number of clusters (default 100), from one through the number of included mask voxels. Note that some methods may produce slightly different numbers of clusters due to algorithmic constraints.

<code>spatial_weight</code>	Method-specific spatial control in $[0, 1]$. For methods that define a convex feature/spatial blend, zero disables the spatial term and one disables the feature term. Other methods use the value as a bounded mapping to a native scale; their endpoints do not have that blend meaning. Default 0.5. This parameter is inactive for "rena" and "rena_plus"; supplying it explicitly for either method is an error. Maps to method-specific parameters: • supervoxels: $\alpha = 1 - \text{spatial_weight}$ (0 = all spatial, 1 = all feature) • snic: $\text{compactness} = \text{spatial_weight} * 10$ (range 0-10) • slic: $\text{compactness} = \text{spatial_weight} * 20$ (typical range 1-20) • corr_slic/brs_slic: direct convex blend between correlation and scaled spatial distance • slice_msf: $\text{compactness} = \text{spatial_weight} * 10$, followed by FH component scale $2 / (\text{compactness} + 1)$. This changes the graph-segmentation scale; it is not a convex spatial/feature blend. • flash3d: $\lambda_s = \text{spatial_weight}$ (direct mapping) • mcl: direct blend between feature and spatial edge similarities
<code>max_iterations</code>	Positive finite integer iteration limit for iterative methods. It is inactive for "snic", "slice_msf", and "commute"; supplying it explicitly for those methods is an error.
<code>connectivity</code>	Integer neighborhood connectivity. Supported values are method-specific: supervoxels accepts 6, 18, 26, or 27; slic, corr_slic, brs_slic, and slice_msf accept 6 or 26; G3S, ReNA variants, and MCL accept 6, 18, or 26. It is inactive for snic, flash3d, acsc, and commute.
<code>parallel</code>	Enable parallel processing for supervoxels and slic. For all other methods, an explicit TRUE is rejected; omitted or FALSE selects the serial implementation. Default TRUE for supported methods.
<code>verbose</code>	Print progress messages. Default FALSE.
<code>sigma1</code>	Bandwidth of heat kernel for features (supervoxels-specific)
<code>sigma2</code>	Bandwidth of heat kernel for coordinates (supervoxels-specific)
<code>use_gradient</code>	Use gradient-based initialization
<code>converge_thresh</code>	Convergence threshold
<code>...</code>	Additional parameters passed to supervoxels

Value

A cluster4d_result object

clustering_accuracy	<i>Compute Clustering Accuracy Metrics</i>
---------------------	--

Description

Compares predicted cluster assignments against ground truth using multiple metrics that are invariant to cluster label permutation.

Usage

```
clustering_accuracy(predicted, truth)
```

Arguments

predicted	Integer vector of predicted cluster assignments
truth	Integer vector of ground truth cluster assignments

Details

The metrics compare the two induced equivalence relations; cluster names do not need to match. All observations are included, including a label of zero. Missing or non-finite labels are rejected rather than silently removed. Computation uses only observed contingency cells and is linear in the number of observations for singleton-heavy partitions.

Adjusted Rand Index (ARI): Measures agreement between two clusterings, adjusted for chance. $ARI = 1$ means perfect agreement, $ARI = 0$ means random, $ARI < 0$ means worse than random.

Variation of information (VI): The sum of the two conditional entropies, measured with base-2 logarithms. VI is zero exactly for identical partitions and increases with disagreement.

Pairwise Dice: $2 * TP / (2 * TP + FP + FN)$ over unordered pairs of observations, where a positive pair is co-clustered. If both partitions contain only singleton clusters, their pairwise Dice is defined as one.

Matched accuracy: The greatest fraction of matched observations under a one-to-one mapping between predicted and truth labels. It is solved exactly on the sparse observed contingency graph. accuracy is retained as an alias for existing callers; it is not the former greedy approximation.

Value

A list with adjusted Rand index (`ari`), variation of information in bits (`variation_of_information_bits`), pairwise Dice (`pairwise_dice`), arithmetic normalized mutual information (`nmi`), exact maximum one-to-one label-matched accuracy (`matched_accuracy`), its compatibility alias `accuracy`, partition counts, and a permutation-invariant perfect flag.

Examples

```
# Perfect clustering
truth <- rep(1:3, each = 10)
pred <- rep(c(2, 3, 1), each = 10) # Same clustering, different labels
clustering_accuracy(pred, truth) # ARI = 1
```

commute_cluster

Commute-time clustering on a physical k-nearest-neighbor graph

Description

Builds an explicitly declared symmetric k-nearest-neighbor graph in physical coordinate space, computes a Laplacian commute-time embedding, and applies deterministically initialized k-means.

Usage

```
commute_cluster(
  bvec,
  mask,
  K = 100,
  ncomp = ceiling(sqrt(K * 2)),
  alpha = 0.5,
  sigma1 = 0.73,
  sigma2 = 5,
  connectivity = 6L,
  weight_mode = c("binary", "heat"),
  bad_voxel_policy = c("error", "zero_constant"),
  variance_tol = 1e-09,
  eigen_tol = 1e-10,
  eigen_solver = c("auto", "dense", "sparse"),
  verbose = FALSE
)
```

Arguments

bvec	A NeuroVec or SparseNeuroVec supplying voxel features.
mask	A NeuroVol mask. Finite values strictly greater than zero are included.
K	Finite integer cluster count from one through the included voxel count.
ncomp	Finite integer embedding dimension from one through N - 1.
alpha	Feature-similarity weight from zero through one.
sigma1	Positive physical-distance heat-kernel bandwidth.
sigma2	Positive feature-distance heat-kernel bandwidth.
connectivity	Number of nearest physical neighbors per directed search. The final undirected graph is the symmetric union of those directed edges; this is not masked grid adjacency and may bridge mask holes.

weight_mode	"binary" or "heat" edge weights.
bad_voxel_policy	"error" (default), or "zero_constant" to map finite zero-variance voxel series to deterministic zero standardized vectors. NA and Inf always fail closed. No noise is injected.
variance_tol	Non-negative threshold defining a constant voxel series.
eigen_tol	Positive tolerance for the Laplacian zero eigenspace.
eigen_solver	"auto" (default), "dense", or "sparse". The dense solver runs a full <code>eigen()</code> on the N-by-N Laplacian: $O(N^2)$ memory and $O(N^3)$ time, practical up to a few thousand voxels. The sparse solver computes only the <code>ncomp + 1</code> smallest eigenpairs by shift-invert Lanczos (<code>eigencore::eig_partial()</code> , a required dependency) and supports larger masks. "auto" uses the dense solver for at most 3,000 masked voxels and the sparse solver above that. Both span the same eigenspace; embeddings agree to solver tolerance, up to the sign of each eigenvector (and the basis inside a repeated eigenvalue), which commute-time distances and hence k-means do not depend on.
verbose	Logical; print progress.

Details

The implementation is deterministic. RNG initialization for the sparse eigensolver and `stats::kmeans()` is scoped to a fixed local seed, after which the caller's prior RNG state (including absence of `.Random.seed`) is restored exactly. With the dense solver it forms a full Laplacian eigendecomposition, requiring quadratic memory and cubic time, so `eigen_solver = "auto"` switches to the sparse solver above 3,000 voxels. Sparse solver memory and time depend on factorization fill-in, the requested embedding dimension, and convergence. Sparse eigenpairs must have original-Laplacian relative residuals at most $1e-10$ and orthogonality error at most $1e-8$. Residuals are scaled by $(\text{norm}(L, "I") + \text{abs}(\lambda)) * \text{norm}(v, "2")$. These checks and `eigencore`'s separate certificate at the solver tolerance are retained in `metadata$spectral$diagnostics`. The sparse solver never silently falls back to a dense solve. A full-spectrum request (`ncomp = N - 1`) uses the dense solver even when `eigen_solver = "sparse"`.

Value

A `commute_time_cluster_result` with contiguous labels, K-by-T original-space feature means, K-by-3 physical coordinate means, the N-by-ncomp embedding, graph and spectral provenance, and explicit data-policy provenance.

compare_cluster4d	<i>Compare multiple cluster4d results</i>
-------------------	---

Description

Compares clustering results over exactly the same included mask voxels and physical coordinate system. Ambiguous center-based scores are not used.

Usage

```
compare_cluster4d(
  ...,
  metrics = c("summary", "spatial_dispersion"),
  feature_mat = NULL
)
```

Arguments

...	cluster4d_result objects to compare
metrics	Character vector selecting explicit estimands: • "summary": cluster-count and cluster-size summaries. • "spatial_dispersion": root mean squared physical distance in millimetres from voxels to their final cluster centroid (lower is more compact). • "temporal_coherence": mean Pearson correlation across all unordered within-cluster voxel pairs, requiring feature_mat. • "partition_agreement": adjusted Rand index, variation of information in bits, and pairwise Dice for exactly two results.
feature_mat	A shared finite numeric matrix with voxels in rows and at least two time/features in columns. Required only for "temporal_coherence". Rows must be non-constant.

Details

Every result must carry canonical mask and affine provenance. The exact included voxel indices and physical geometry are checked before any cross-result comparison. Partition metrics use only observed contingency cells and never construct an N-by-N co-membership matrix. Pairwise agreement columns are repeated on the two result rows so the return value remains a data frame.

Value

A data frame with one row per result and explicitly named metrics.

compress_features_svd *Compress High-Dimensional Features via Randomized SVD*

Description

Reduces the dimensionality of fMRI time series data using Singular Value Decomposition (SVD) while preserving the majority of variance. This is a key component of the G3S (Gradient-Guided Geodesic Supervoxels) algorithm, enabling 10-20x speedup in similarity computations.

Usage

```
compress_features_svd(
  feature_mat,
  n_components = 15,
  variance_threshold = 0.95,
  use_irlba = TRUE,
  use_rsvd = TRUE
)
```

Arguments

feature_mat	A numeric matrix with voxels in rows and timepoints in columns (N x T). The function will center and scale the data automatically.
n_components	Target number of dimensions (default: 15). Higher values preserve more variance but reduce speed gains.
variance_threshold	Minimum proportion of variance to preserve (0-1). If the requested n_components doesn't meet this threshold, more components will be retained. Default: 0.95 (95% variance).
use_irlba	Logical; if TRUE and the irlba package is available, use fast randomized SVD for large datasets (>10,000 voxels). Default: TRUE.
use_rsvd	Logical; if TRUE and the rsvd package is available, prefer the randomized SVD implementation from rsvd for large datasets (>10,000 voxels). This can outperform irlba on tall-and-skinny matrices. Default: TRUE.

Details**Algorithm:**

The compression follows these steps:

1. **Center and scale:** Each timepoint is centered (mean=0) and scaled (sd=1) to ensure equal contribution from all timepoints.
2. **SVD decomposition:** Computes $X = U * D * t(V)$ where:
 - U (N x k): Left singular vectors (voxel loadings)
 - D (k x k): Diagonal matrix of singular values
 - V (T x k): Right singular vectors (time loadings)
3. **Compression:** Retains only the first M components where M is chosen to balance speed (fewer components) and accuracy (more variance explained).
4. **Normalization:** Each compressed feature vector is normalized to unit length, enabling fast cosine similarity via dot products: $\text{cor}(x, y) \approx x * y$

Performance Characteristics:

- **Memory:** Compressed data uses M/T of original size (e.g., 15/300 = 5%)
- **Speed:** Similarity calculations are M/T times faster (e.g., 20x for 15/300)
- **Accuracy:** Typically preserves 95%+ of signal with M=10-20 for fMRI data

Choosing n_components:

- **Small (M=10):** Maximum speed, good for noisy or heavily smoothed data
- **Medium (M=15):** Balanced, recommended default for most fMRI data
- **Large (M=25):** High accuracy, use for high SNR data or critical applications

The function will automatically increase `n_components` if needed to meet `variance_threshold`. If that requires the full available rank, the exact full-rank decomposition is used.

Value

A list with components:

<code>features</code>	Compressed feature matrix (N x M), starting from the requested <code>n_components</code> and increasing M when needed to satisfy <code>variance_threshold</code> . Each row is normalized to unit length for cosine similarity calculations.
<code>variance_explained</code>	Proportion of total variance explained by the retained components (0-1).
<code>n_components</code>	Actual number of components retained (may be > <code>n_components</code> input if needed to meet <code>variance_threshold</code>).
<code>rotation</code>	The right singular vectors (V matrix) for transforming new data.
<code>singular_values</code>	The singular values (d vector) for each component.
<code>center</code>	Column means used for centering (for transforming new data).
<code>scale</code>	Column standard deviations used for scaling (for transforming new data).

Reproducibility

Both randomized backends draw random projections. They are run under a fixed internal seed and the caller's RNG state is restored afterwards, so repeated calls on the same data return the same decomposition and the function never perturbs the surrounding stream. Without that, two identical `cluster4d_g3s` runs could disagree on half their voxel labels.

See Also

`cluster4d_g3s` for the full G3S clustering algorithm. `transform_new_data_svd` for applying the compression to new data.

Examples

```
# Simulate fMRI time series data
n_voxels <- 1000
n_timepoints <- 300
latent <- matrix(rnorm(n_voxels * 8), n_voxels, 8)
loadings <- matrix(rnorm(n_timepoints * 8), n_timepoints, 8)
feature_mat <- latent %*% t(loadings) +
  matrix(rnorm(n_voxels * n_timepoints, sd = 0.05), n_voxels, n_timepoints)

# Compress to 15 dimensions (typical for G3S)
compressed <- compress_features_svd(feature_mat, n_components = 15)
print(compressed$variance_explained) # At least 0.95
```

```

print(ncol(compressed$features))      # 15 here; may increase when necessary

# More aggressive compression
compressed_fast <- compress_features_svd(feature_mat, n_components = 10)

# Conservative compression (preserve 98% variance)
compressed_accurate <- compress_features_svd(
  feature_mat,
  n_components = 15,
  variance_threshold = 0.98
)

```

compute_centroids	<i>Compute Centroids of Clusters</i>
-------------------	--------------------------------------

Description

The `compute_centroids` function calculates the center and centroid of each cluster given a feature matrix, grid, and cluster assignments.

Usage

```
compute_centroids(feature_mat, grid, assignment, medoid = FALSE)
```

Arguments

<code>feature_mat</code>	A matrix of features, where columns represent data points and rows represent features.
<code>grid</code>	A matrix representing the spatial grid of data points.
<code>assignment</code>	A vector containing the cluster assignment for each data point.
<code>medoid</code>	A logical value indicating whether to calculate medoids instead of means for cluster centers and centroids. Default is FALSE.

Value

A list containing two elements:

<code>center</code>	A matrix containing the centers of each cluster.
<code>centroid</code>	A matrix containing the centroids of each cluster.

Examples

```

# Simple synthetic example
feature_mat <- matrix(rnorm(100), nrow = 10, ncol = 10)
grid <- cbind(x = runif(10), y = runif(10), z = runif(10))
assignment <- rep(1:2, each = 5)
centroids <- compute_centroids(feature_mat, grid, assignment)

```

```

print(names(centroids))

## Not run:
# Larger example with real neuroimaging data
# Assuming `feature_mat`, `grid`, and `assignment` are available
centroids <- compute_centroids(feature_mat, grid, assignment)
# To compute medoids instead of means
medoids <- compute_centroids(feature_mat, grid, assignment, medoid=TRUE)

## End(Not run)

```

compute_functional_gradient

Compute Functional Gradient from Feature Matrix

Description

Calculates the functional gradient for each voxel as the average dissimilarity to its spatial neighbors in feature space. Low gradient values indicate stable, homogeneous regions (good seed locations); high values indicate boundaries.

Usage

```
compute_functional_gradient(feature_mat, coords, k_neighbors = 26)
```

Arguments

feature_mat	Numeric matrix (N x M) of features (voxels x dimensions).
coords	Numeric matrix (N x 3) of spatial coordinates.
k_neighbors	Integer; number of nearest spatial neighbors to use. Default: 26.

Details

For each voxel i , the functional gradient is:

$$grad(i) = \frac{1}{k} \sum_{j \in N_k(i)} (1 - f_i \cdot f_j)$$

where $N_k(i)$ are the k nearest spatial neighbors and $f_i \cdot f_j$ is the dot product (cosine similarity for unit-length vectors).

Value

Numeric vector of length N containing gradient values for each voxel.

Examples

```
## Not run:
features <- matrix(rnorm(500 * 15), 500, 15)
features <- t(apply(features, 1, function(x) x / sqrt(sum(x^2))))
coords <- matrix(rnorm(500 * 3), 500, 3)

grad <- compute_functional_gradient(features, coords, k_neighbors = 26)
hist(grad, breaks = 50, main = "Functional Gradient Distribution")

## End(Not run)
```

detrend_basis_cpp	<i>Flexible Detrending with Basis Functions (Fast C++ Implementation)</i>
-------------------	---

Description

Removes trends from each voxel's timeseries by projecting out a basis. Supports polynomial or DCT (discrete cosine transform) basis functions.

Usage

```
detrend_basis_cpp(data, basis)
```

Arguments

data	Numeric matrix with voxels as rows and timepoints as columns
basis	Orthonormal basis matrix (n_time x n_basis) to project out. Use <code>make_dct_basis()</code> or <code>make_poly_basis()</code> to generate.

Details

For orthonormal basis B , computes residuals as: $y_{\text{detrend}} = y - B * (B^T * y)$

This is equivalent to regressing out the basis and keeping residuals.

Value

Numeric matrix with basis components removed from each row

Examples

```
## Not run:
# Remove linear + quadratic trend (polynomial degree 2)
basis <- make_poly_basis(n_time = 100, degree = 2)
data_detrend <- detrend_basis_cpp(data, basis)

# Remove low-frequency drift with DCT (first 5 components)
basis <- make_dct_basis(n_time = 100, n_basis = 5)
```

```
data_detrend <- detrend_basis_cpp(data, basis)

## End(Not run)
```

detrend_dct_cpp	<i>DCT Detrending (Convenience Function)</i>
-----------------	--

Description

Removes low-frequency components using discrete cosine transform.

Usage

```
detrend_dct_cpp(data, n_basis = 4L)
```

Arguments

data	Numeric matrix with voxels as rows and timepoints as columns
n_basis	Number of DCT basis functions to remove (including constant). Default 4 removes constant, linear, and first two cosine harmonics.

Details

DCT detrending is commonly used in fMRI preprocessing. The number of basis functions controls the high-pass filter cutoff:

- n_basis = 1: mean removal only
- n_basis = 2: ~ linear detrend
- n_basis = 4-6: typical for fMRI (removes drift < ~0.01 Hz for TR=2s)

Higher n_basis = more aggressive high-pass filtering.

Value

Numeric matrix with low-frequency components removed

detrend_poly_cpp	<i>Polynomial Detrending (Convenience Function)</i>
------------------	---

Description

Removes polynomial trend up to specified degree from each voxel.

Usage

```
detrend_poly_cpp(data, degree = 1L)
```

Arguments

data	Numeric matrix with voxels as rows and timepoints as columns
degree	Polynomial degree (0 = mean, 1 = linear, 2 = quadratic, etc.)

Details

Convenience wrapper that generates polynomial basis and calls `detrend_basis_cpp`. For `degree=1`, this is equivalent to `detrend_time_cpp` but slightly slower.

Value

Numeric matrix with polynomial trend removed

detrend_time	<i>Detrend Voxel Timeseries</i>
--------------	---------------------------------

Description

Fast removal of trends from each voxel's timeseries in 4D neuroimaging data. Supports linear, polynomial, and DCT (discrete cosine transform) detrending.

Usage

```
detrend_time(
  vec,
  mask = NULL,
  method = c("linear", "poly", "dct"),
  degree = 2,
  n_basis = 4
)
```

Arguments

vec	A NeuroVec object (4D neuroimaging data)
mask	Optional NeuroVol mask. If NULL, uses all non-zero voxels.
method	Detrending method: "linear" (default), "poly", or "dct"
degree	For method="poly", the polynomial degree (1=linear, 2=quadratic, etc.)
n_basis	For method="dct", number of DCT basis functions to remove. Higher values = more aggressive high-pass filtering.

Details

Three detrending methods are available:

Linear (method="linear"): Fastest option. Removes linear drift by fitting $y = a + b \cdot t$.

Polynomial (method="poly"): Removes polynomial trend up to specified degree. degree=1 is linear, degree=2 adds quadratic, etc. Uses orthonormalized basis for stability.

DCT (method="dct"): Removes low-frequency components using discrete cosine transform basis. This is a standard high-pass filter for fMRI data. The n_basis parameter controls the cutoff:

- n_basis=1: mean removal only
- n_basis=2: approximately linear detrend
- n_basis=4-6: typical for fMRI (removes ~0.01 Hz drift for TR=2s)

All methods use fast C++ parallel implementation via RcppParallel.

Value

A NeuroVec with trend removed from each voxel

Examples

```
## Not run:
vec <- neuroim2::read_vec("functional.nii.gz")

# Linear detrend (fastest)
vec_lin <- detrend_time(vec)

# Quadratic detrend
vec_quad <- detrend_time(vec, method = "poly", degree = 2)

# DCT high-pass filter (common for fMRI)
vec_dct <- detrend_time(vec, method = "dct", n_basis = 5)

## End(Not run)
```

detrend_time_cpp	<i>Detrend Voxel Timeseries (Fast C++ Implementation)</i>
------------------	---

Description

Removes linear trend from each voxel's timeseries using fast parallel linear regression.

Usage

```
detrend_time_cpp(data)
```

Arguments

data	Numeric matrix with voxels as rows and timepoints as columns
------	--

Details

For each voxel, fits a linear model $y = \text{intercept} + \text{slope} * t$ and subtracts the fitted values: residuals = $y - \text{fitted}$.

Uses RcppParallel for fast parallel computation across voxels. The linear regression is computed efficiently using precomputed time statistics.

Value

Numeric matrix with linear trend removed from each row (voxel)

er_ml_rena	<i>Backwards-compatible alias for rena_plus</i>
------------	---

Description

Backwards-compatible alias for rena_plus

Usage

```
er_ml_rena(bvec, mask, ...)
```

Arguments

bvec	A NeuroVec providing voxel time-series.
mask	A NeuroVol mask; finite positive voxels are included.
...	Reserved for future options.

Value

Same as [rena_plus](#).

`estimate_optimal_components`*Estimate Optimal Number of SVD Components*

Description

Uses the elbow method or cumulative variance to suggest an appropriate number of components for SVD compression.

Usage

```
estimate_optimal_components(  
  feature_mat,  
  max_components = 50,  
  method = c("variance", "elbow"),  
  variance_target = 0.95  
)
```

Arguments

<code>feature_mat</code>	Numeric matrix (N x T)
<code>max_components</code>	Maximum components to test (default: 50)
<code>method</code>	Either "elbow" (find elbow in scree plot) or "variance" (return number needed for 95% variance). Default: "variance".
<code>variance_target</code>	Target variance for "variance" method (default: 0.95)

Value

Integer; suggested number of components

Examples

```
## Not run:  
data <- matrix(rnorm(1000 * 300), 1000, 300)  
n_opt <- estimate_optimal_components(data)  
print(n_opt)  
  
## End(Not run)
```

find_gradient_seeds_g3s

Find G3S Seeds via Functional Gradient Minima

Description

Identifies optimal seed locations for G3S clustering by finding local minima in the functional gradient field. Unlike uniform grid seeding, this approach places seeds in the centers of stable functional regions rather than on boundaries.

Usage

```
find_gradient_seeds_g3s(
    feature_mat,
    coords,
    K,
    k_neighbors = 26,
    oversample_ratio = 3,
    min_separation_factor = 1,
    distance = c("cosine", "euclidean"),
    knn = NULL,
    spatial_scale = NULL
)
```

Arguments

feature_mat	Numeric matrix (N x M) of compressed features, where each row is a voxel and each column is a feature dimension. Should be normalized to unit length for cosine similarity (as output by compress_features_svd).
coords	Numeric matrix (N x 3) of spatial coordinates for each voxel, typically in mm units from <code>neuroim2::index_to_coord()</code> .
K	Integer; target number of seeds (clusters). The function may return fewer seeds if spatial separation constraints cannot be satisfied.
k_neighbors	Integer; number of nearest neighbors to use for gradient computation. Default: 26 (full 3D connectivity).
oversample_ratio	Numeric; ratio of candidates to K considered before spatial separation is applied. Default: 3. Candidates are the lowest-gradient voxels <i>within each spatial cell</i> rather than the globally lowest, so a flat or heavily tied gradient field cannot concentrate the whole pool in one part of the mask.
min_separation_factor	Numeric; starting minimum separation between seeds, as a multiple of the expected supervoxel scale. Default: 1. This is a starting point only: the radius decays until K seeds fit, so the outcome is the widest separation the mask actually admits and is insensitive to the exact starting value.

distance	Character; distance metric for gradient computation. One of "cosine" (default) or "euclidean".
knn	Optional pre-computed k-nearest-neighbor object. If NULL (default), KNN is computed internally.
spatial_scale	Optional positive physical length scale for the seed inhibition radius, normally the masked-volume scale used for compactness. When NULL the radius is estimated from the coordinate bounding box, which can overestimate it on non-convex masks.

Details

Algorithm:

The seeding process follows these steps:

1. **Functional Gradient Computation:** For each voxel, compute the average dissimilarity to its k nearest spatial neighbors in feature space:

$$grad(i) = \frac{1}{k} \sum_{j \in N(i)} dist(f_i, f_j)$$

where $dist(f_i, f_j) = 1 - f_i \cdot f_j$ (cosine distance).

2. **Candidate Selection:** Rank voxels by gradient value and select the top $K * oversample_ratio$ voxels with lowest gradient (most stable regions).
3. **Spatial Separation:** Starting with the lowest-gradient candidate, greedily select seeds that maintain minimum spatial separation from all previously selected seeds. If that pool cannot provide K seeds at the physical inhibition radius, deterministic farthest-point sampling over the full mask fills the shortfall.

Why Gradient-Based Seeding?:

Traditional uniform grid seeding often places seeds on functional boundaries (high gradient regions), leading to:

- Poor initial cluster centroids
- Slower convergence
- Less biologically plausible parcellations

Gradient-based seeding ensures:

- Seeds are in functional cores (low gradient = stable, homogeneous regions)
- Better initial centroids = faster, more accurate clustering
- Biologically plausible: aligns with cortical organization

Comparison with Other Methods:

- **vs. Uniform Grid:** G3S gradient seeding reduces boundary artifacts and improves convergence speed by 2-3x.
- **vs. K-means Initialization:** G3S respects spatial structure, preventing disconnected clusters.
- **vs. Random Seeding:** G3S is deterministic given the data, improving reproducibility.

Value

Integer vector of length K containing the row indices of selected seed voxels in the feature_mat/coords matrices.

See Also

[compute_functional_gradient](#) for gradient computation details. [cluster4d_g3s](#) for the full G3S clustering algorithm. [spatial_gradient](#) for spatial-only gradient (used in SNIC).

Examples

```
## Not run:
# Simulate compressed fMRI features
n_voxels <- 1000
n_components <- 15
features <- matrix(rnorm(n_voxels * n_components), n_voxels, n_components)
features <- t(apply(features, 1, function(x) x / sqrt(sum(x^2)))) # Normalize

# Spatial coordinates (10x10x10 grid)
coords <- as.matrix(expand.grid(x = 1:10, y = 1:10, z = 1:10))

# Find 20 seeds
seeds <- find_gradient_seeds_g3s(features, coords, K = 20)
print(length(seeds)) # Should be 20 or close to it

# Visualize gradient values
grad_vals <- compute_functional_gradient(features, coords)
print(summary(grad_vals[seeds])) # Should be low values

## End(Not run)
```

find_initial_points	<i>Find Initial Cluster Centers for Supervoxel Algorithm (Deprecated)</i>
---------------------	---

Description

Deprecated: This function is now available as `find_gradient_seeds()` in `cluster4d_init.R`. This version is maintained for backward compatibility.

This function finds the initial cluster centers for a supervoxel algorithm. Supervoxels are used to partition 3D image data into volumetric regions, grouping similar voxels together. The initial cluster centers are crucial for the performance and quality of the final supervoxels.

Usage

```
find_initial_points(cds, grad, K = 100)
```

Arguments

<code>cds</code>	A matrix or data frame representing the spatial coordinates of the voxels.
<code>grad</code>	A vector representing the gradient values of the voxels.
<code>K</code>	The desired number of supervoxels (clusters) in the output (default: 100).

Value

A list containing two elements: `selected` - a vector of the selected indices corresponding to the initial cluster centers, `coords` - a matrix or data frame with the spatial coordinates of the initial cluster centers.

See Also

[find_gradient_seeds](#), [initialize_clusters](#)

`generate_synthetic_volume`

Generate a synthetic 4D neuroimaging volume with labeled clusters

Description

Convenient synthetic data for examples, vignettes, and tests. Each scenario produces a `NeuroVec` (time series volume), a `NeuroVol` mask, and the ground-truth cluster labels so you can benchmark algorithms or build illustrations quickly.

Usage

```
generate_synthetic_volume(
  scenario = c("gaussian_blobs", "z_layers", "checkerboard"),
  dims = c(16, 16, 8),
  n_clusters = 4,
  n_time = 30,
  spread = c(3.5, 3.5, 2),
  noise_sd = 0.05,
  amplitude_range = c(0.8, 1.2),
  spacing_mm = c(3, 3, 3),
  pattern_type = c("orthogonal", "harmonic", "random"),
  seed = NULL
)
```

Arguments

<code>scenario</code>	Pattern to embed. One of "gaussian_blobs", "z_layers", or "checkerboard".
<code>dims</code>	Integer vector of length 3 giving the spatial grid dimensions.
<code>n_clusters</code>	Number of latent clusters to embed.

n_time	Number of time points.
spread	Characteristic spread of each Gaussian blob (scalar or length 3).
noise_sd	Standard deviation of additive Gaussian noise.
amplitude_range	Range used when scaling each voxel's amplitude.
spacing_mm	Physical voxel size supplied to the generated NeuroSpace.
pattern_type	Type of temporal patterns to generate. One of "orthogonal" (QR-decomposition for maximally distinct signals, default), "harmonic" (phase/frequency-shifted sinusoids), or "random" (random linear combinations of harmonics).
seed	Optional seed for reproducibility.

Value

A list with elements vec, mask, truth, coords, patterns, weights, dims, n_clusters, and scenario.

Examples

```
syn <- generate_synthetic_volume(
  scenario = "gaussian_blobs",
  dims = c(12, 12, 6),
  n_clusters = 4,
  seed = 1
)
str(syn)
truth_vol <- array(syn$truth, syn$dims)
clusvol <- neuroim2::ClusteredNeuroVol(syn$mask > 0, clusters = syn$truth)
# Visualise a single slice (requires neuroim2 plot method):
## Not run:
plot(clusvol)

## End(Not run)
```

is.cl_partition.cluster_result

Test if Object is a Partition

Description

This function tests whether a cluster result object represents a partition. It is a method for the [is.cl_partition](#) generic from the clue package. For cluster_result objects, this always returns TRUE since cluster results represent valid partitions where each data point belongs to exactly one cluster.

Usage

```
## S3 method for class 'cluster_result'
is.cl_partition(x)
```

Arguments

x A cluster_result object.

Value

TRUE, indicating that cluster results are always valid partitions.

See Also

[is.cl_partition](#) for the generic function.

knn_shrink	<i>K-nearest-neighbor shrink</i>
------------	----------------------------------

Description

Replace each voxel by the mean of its k nearest neighbors in its local spatial neighborhood.

Usage

```
knn_shrink(bvec, mask, k = 5, connectivity = 27)
```

Arguments

bvec A [NeuroVec](#) instance (the data).
 mask A [NeuroVol](#) mask defining the voxels to include. If numeric, nonzero = included.
 k The number of nearest neighbors to average over.
 connectivity The number of spatial neighbors to include in the search around each voxel.

Value

A SparseNeuroVec or similar object with the smoothed data.

Examples

```
## Not run:
mask <- neuroim2::NeuroVol(array(1, c(20,20,20)), neuroim2::NeuroSpace(c(20,20,20)))
bvec <- replicate(10,
  neuroim2::NeuroVol(array(runif(20*20*20), c(20,20,20)),
    neuroim2::NeuroSpace(c(20,20,20))),
  simplify=FALSE)
bvec <- do.call(neuroim2::concat, bvec)
sbvec <- knn_shrink(bvec, mask, k=3)

## End(Not run)
```

make_block_synthetic *Make a tiny block-structured synthetic volume*

Description

A scikit-learn style make_blobs for neuroimaging demos. Builds a small 2D/3D mask with three spatial bands (left/mid/right) and clearly separated time-series patterns plus mild Gaussian noise. Designed to be fast, deterministic, and visually intuitive in vignettes and tests.

Usage

```
make_block_synthetic(
    dims = c(12, 12, 1),
    ntime = 80,
    noise = 0.15,
    seed = NULL
)
```

Arguments

dims	Integer vector length 3, spatial dimensions. Default c(12, 12, 1).
ntime	Number of time points. Default 80.
noise	Standard deviation of added Gaussian noise. Default 0.15.
seed	Optional integer seed for reproducibility. Default NULL (no change).

Value

List with components:

vec NeuroVec 4D volume

mask NeuroVol mask

truth Integer ground-truth labels (length = prod(dims))

make_dct_basis *Generate DCT-II Basis Matrix*

Description

Creates a discrete cosine transform (type II) basis matrix for detrending.

Usage

```
make_dct_basis(n_time, n_basis)
```

Arguments

n_time	Number of timepoints
n_basis	Number of DCT basis functions (including constant)

Details

DCT basis functions are excellent for removing low-frequency drift in fMRI. The first basis is constant (mean), subsequent bases capture increasingly higher frequency components.

Value

Numeric matrix (n_time x n_basis) with orthonormal DCT basis

make_poly_basis	<i>Generate Polynomial Basis Matrix</i>
-----------------	---

Description

Creates a polynomial basis matrix for detrending.

Usage

make_poly_basis(n_time, degree)

Arguments

n_time	Number of timepoints
degree	Polynomial degree (0 = constant, 1 = linear, 2 = quadratic, etc.)

Details

Uses Gram-Schmidt orthonormalization for numerical stability. Degree 0 = mean removal, 1 = linear detrend, 2 = quadratic, etc.

Value

Numeric matrix (n_time x (degree+1)) with orthonormalized polynomial basis

make_synthetic_clusters

Create Synthetic Ground Truth Clustering Data

Description

Generates a volume with perfectly separable clusters for testing clustering algorithm correctness. Each cluster is a spatially contiguous cubic sub-region with identical timeseries within the cluster and orthogonal signals between clusters.

Usage

```
make_synthetic_clusters(
    n_time = 100,
    noise_sd = 0,
    n_clusters = 27,
    dim_per_cluster = 3,
    seed = NULL
)
```

Arguments

n_time	Number of timepoints. Default 100.
noise_sd	Standard deviation of Gaussian noise to add. Default 0 (no noise = perfect separability).
n_clusters	Number of clusters. Must be a perfect cube (8, 27, 64, etc.) for cubic cluster geometry. Default 27.
dim_per_cluster	Voxels per dimension per cluster. Default 3.
seed	Random seed for reproducibility. Default NULL.

Details

For the default n_clusters=27, creates a 9x9x9 volume divided into 27 cubic 3x3x3 clusters arranged in a 3x3x3 grid:

z=1-3:	z=4-6:	z=7-9:
y ^	y ^	y ^
7 8 9	16 17 18	25 26 27
4 5 6	13 14 15	22 23 24
1 2 3	10 11 12	19 20 21
+-----> x	+-----> x	+-----> x

Each cluster has a unique orthogonal signal generated from QR decomposition of a random matrix, ensuring zero correlation between clusters.

This creates an "easy" test case where any reasonable clustering algorithm should achieve perfect or near-perfect recovery with noise_sd=0.

Value

A list with:

vec	NeuroVec object
mask	NeuroVol mask (all ones)
true_clusters	NeuroVol with ground truth cluster labels
true_labels	Integer vector of true cluster assignments
n_clusters	Number of clusters
cluster_size	Size of each cluster in voxels
signals	Matrix of cluster signals (n_time x n_clusters)

Examples

```
# Default: 27 clusters, perfect separability
data <- make_synthetic_clusters()
table(data$true_labels) # 27 voxels per cluster

# 8 clusters (2x2x2 grid of 3x3x3 cubes = 6x6x6 volume)
data <- make_synthetic_clusters(n_clusters = 8)

# With noise for more realistic testing
data <- make_synthetic_clusters(noise_sd = 0.5)
```

merge_clus	<i>Merge Clustering Results Using a Consensus Clustering Algorithm</i>
------------	--

Description

The merge_clus function combines a set of clustering results using a consensus clustering algorithm.

Usage

```
merge_clus(x, ...)
```

Arguments

x	A clustering result, typically a list or an object of class "cluster_result".
...	Additional clustering results to be merged and method-specific arguments (e.g., method for consensus algorithm selection).

Value

A `ClusteredNeuroVol` instance.

See Also

[cl_consensus](#), [as.cl_hard_partition](#), [cl_ensemble](#)

Examples

```
## Not run:
# Assuming clustering1, clustering2, and clustering3 are objects of class "cluster_result"
merged_clustering <- merge_clus(clustering1, clustering2, clustering3, method="SE")

## End(Not run)
```

```
merge_clus.cluster_result
```

Merge Clustering Results for ClusteredNeuroVol Objects

Description

This method of `merge_clus` is specifically designed to merge clustering results for `ClusteredNeuroVol` objects by performing consensus clustering across multiple clustering results.

Usage

```
## S3 method for class 'cluster_result'
merge_clus(x, method = "SE", ...)
```

Arguments

<code>x</code>	A <code>ClusteredNeuroVol</code> object or an object of class <code>"cluster_result"</code> .
<code>method</code>	A character string indicating the consensus clustering algorithm to use. Default is <code>"SE"</code> . See cl_consensus for available methods.
<code>...</code>	Additional clustering results to be merged.

Value

A [ClusteredNeuroVol](#) instance representing the consensus partition.

See Also

[cl_consensus](#), [as.cl_hard_partition](#), [cl_ensemble](#)

meta_clust

Meta Clustering for Cluster Results (S4 Generic)

Description

The meta_clust generic function is used to define methods for different classes of input objects.

Usage

```
meta_clust(x, cuts, ...)
```

Arguments

x	A clustering result, typically an object of class "cluster_result".
cuts	The number of cluster cuts to consider.
...	Additional arguments passed to methods.

Value

Depends on the method called.

meta_clust.cluster_result

Meta Clustering for Cluster Results

Description

The meta_clust function performs meta clustering on a given clustering result by applying hierarchical clustering on the cluster centers.

Usage

```
## S3 method for class 'cluster_result'
meta_clust(x, cuts = NULL, ...)
```

Arguments

x	A clustering result, typically an object of class "cluster_result".
cuts	Integer vector specifying the number of cluster cuts to consider. Default is NULL, which generates cuts at 2, 5, and 10 clusters (or fewer depending on the number of input clusters).
...	Additional arguments: algo A character string indicating the clustering algorithm to use. Default is "hclust" (hierarchical clustering).

dist_method Character string: "correlation" (default) or "euclidean" for distance calculation between cluster centers.

hclust_method A character string specifying the agglomeration method to use for hierarchical clustering. Default is "ward.D".

Details

Parallelization Status:

Currently NOT parallelized. Meta-clustering performs hierarchical clustering on cluster centers using sequential R functions.

Sequential Operations::

1. **Distance Matrix Computation:**

- Computes correlation distance ($1 - \text{cor}$) between all cluster centers
- $O(K^2)$ pairwise correlations where K = number of input clusters
- Sequential double loop in base R

2. **Hierarchical Clustering:** `hclust()`

- Standard hierarchical agglomeration (Ward.D by default)
- Sequential merging of closest clusters
- $O(K^2 \log K)$ complexity

3. **Tree Cutting:** `cutree()`

- Cuts dendrogram at multiple heights
- Creates nested cluster assignments
- Linear in number of clusters

Why Not Parallelized::

- **Small scale:** Usually operates on hundreds of clusters, not thousands of voxels
- **R built-ins:** Uses standard `hclust()` which is sequential
- **Fast enough:** Typically completes in seconds even for large K
- **Memory efficient:** Only stores $K \times K$ distance matrix

Potential for Parallelization::

- Distance matrix computation could use parallel correlation
- Some hierarchical clustering variants support parallelization
- Multiple cuts could be computed in parallel

Performance Characteristics::

- **Input size:** K clusters from initial clustering (typically 100-1000)
- **Complexity:** $O(K^2)$ for distances, $O(K^2 \log K)$ for clustering
- **Memory:** $O(K^2)$ for distance matrix
- **Speed:** Usually < 1 second for $K < 500$

Performance Tips::

- **Pre-reduce clusters:** Start with fewer initial clusters if meta-clustering is slow
- **Use appropriate linkage:** Ward.D is slower but often gives better results
- **Limit cuts:** Fewer cut levels = faster processing
- **Alternative:** Use consensus clustering (`merge_clus()`) for different approach

Use Cases::

Meta-clustering is useful for:

- Creating multi-resolution parcellations
- Hierarchical organization of functional regions
- Reducing large numbers of clusters to interpretable groups
- Finding stable cluster boundaries across scales

Value

A list containing:

cvol	A list of ClusteredNeuroVol instances.
cuts	The number of cluster cuts.
cutmat	A matrix representing the cluster assignments for each cut.
hclus	The hierarchical clustering result.
original_result	The original clustering result (optional reference).

See Also

[hclust](#), [cutree](#)

neurocluster_simd_info
<i>SIMD / CPU Feature Report (Development Helper)</i>

Description

Reports build-time SIMD feature macros and (when available) a small set of runtime CPU feature flags, plus the currently-selected int8 dot-product kernel used by neurocluster’s correlation-SLIC code paths.

Usage

neurocluster_simd_info()

Details

This is primarily a development/benchmarking aid. For definitive evidence that an ISA-specific kernel is in use, disassemble the package shared library and look for the corresponding instructions.

Value

A named list with fields compiler, arch, compile, runtime, dot_i8_kernel, shared_library, and notes.

normalize_detrend	<i>Normalize and Detrend Neuroimaging Data</i>
-------------------	--

Description

Combined fast preprocessing: removes volume mean offsets and linear trends in a single efficient operation.

Usage

```
normalize_detrend(vec, mask = NULL)
```

Arguments

vec	A NeuroVec object (4D neuroimaging data)
mask	Optional NeuroVol mask. If NULL, uses all non-zero voxels.

Details

Combines [normalize_volumes](#) and [detrend_time](#) in a single pass for efficiency. The operations are applied in order:

1. Remove volume-wise mean offset
2. Remove linear trend from each voxel's timeseries

Uses fast C++ parallel implementation via RcppParallel.

Value

A NeuroVec with volume offsets and linear trends removed

Examples

```
## Not run:  
vec <- neuroim2::read_vec("functional.nii.gz")  
vec_clean <- normalize_detrend(vec)  
  
## End(Not run)
```

normalize_detrend_cpp	<i>Normalize Volumes and Detrend Timeseries (Fast C++ Implementation)</i>
-----------------------	---

Description

Combined operation: first removes volume mean offsets, then removes linear trend from each voxel's timeseries. More efficient than calling both functions separately.

Usage

```
normalize_detrend_cpp(data)
```

Arguments

data	Numeric matrix with voxels as rows and timepoints as columns
------	--

Details

Performs two operations in a single parallel pass:

1. Centers each volume to zero mean
2. Removes linear trend from each voxel's (now centered) timeseries

Value

Numeric matrix with volume offsets and linear trends removed

normalize_volumes	<i>Normalize Volumes by Removing Mean Offset</i>
-------------------	--

Description

Fast removal of volume-wise mean offset from 4D neuroimaging data. Each volume/timepoint is centered to have zero mean across voxels.

Usage

```
normalize_volumes(vec, mask = NULL)
```

Arguments

vec	A NeuroVec object (4D neuroimaging data)
mask	Optional NeuroVol mask. If NULL, uses all non-zero voxels.

Details

This function removes global signal offset that can vary across timepoints due to scanner drift, physiological noise, or other factors. For each timepoint t :

$$output[:, t] = input[:, t] - mean(input[:, t])$$

Uses fast C++ parallel implementation via RcppParallel.

Value

A NeuroVec with each volume centered to zero mean

Examples

```
## Not run:
vec <- neuroim2::read_vec("functional.nii.gz")
vec_normalized <- normalize_volumes(vec)

## End(Not run)
```

normalize_volumes_cpp *Normalize Volumes by Removing Mean Offset (Fast C++ Implementation)*

Description

Removes the mean offset from each volume/timepoint in a matrix. This centers each volume to have zero mean across voxels.

Usage

```
normalize_volumes_cpp(data)
```

Arguments

data Numeric matrix with voxels as rows and timepoints as columns

Details

For each timepoint t , computes $mean_t = mean(data[, t])$ and subtracts it from all voxels: $output[, t] = data[, t] - mean_t$

Uses RcppParallel for fast parallel computation across voxels.

Value

Numeric matrix with each column (volume) centered to mean zero

```
plot.cluster4d_result Plot cluster4d result
```

Description

Creates visualization of clustering results. Shows axial, sagittal, and coronal slices through the clustered volume.

Usage

```
## S3 method for class 'cluster4d_result'
plot(x, slice = NULL, view = "all", colors = NULL, zlim = NULL, ...)
```

Arguments

x	A cluster4d_result object
slice	Slice specification. Can be: • NULL (default): Shows middle slices • Numeric vector of length 3: c(x, y, z) coordinates • "montage": Shows multiple slices
view	Viewing plane: "axial", "sagittal", "coronal", or "all"
colors	Color palette for clusters. Default uses rainbow colors.
zlim	Numeric length-two label range shared by every displayed plane. The default, c(0.5, x\$n_clusters + 0.5), keeps each positive integer label mapped to the same categorical color even when a slice omits labels.
...	Additional arguments passed to plotting functions

Value

Invisibly returns the plotted data

```
print.cluster4d_result
S3 Methods for cluster4d_result Objects
```

Description

Methods for printing, summarizing, and plotting cluster4d results. Print cluster4d result

Usage

```
## S3 method for class 'cluster4d_result'
print(x, ...)
```

Arguments

x	A cluster4d_result object
...	Additional arguments (ignored)

Value

Invisibly returns x

print.g3s_result	<i>Print Method for G3S Results</i>
------------------	-------------------------------------

Description

Print Method for G3S Results

Usage

```
## S3 method for class 'g3s_result'
print(x, ...)
```

Arguments

x	A g3s_result object
...	Additional arguments (ignored)

randomize_cluster_ids	<i>Randomize Cluster IDs</i>
-----------------------	------------------------------

Description

Randomly permute cluster IDs to break any spatial ordering. This is useful when visualizing clusters with continuous color scales, as spatially-ordered cluster IDs create artificial gradients that can be misleading.

Usage

```
randomize_cluster_ids(cluster_result, seed = NULL)
```

Arguments

cluster_result	A clustering result object (e.g., from cluster4d, supervoxels_flash3d, slice_msf, etc.) containing at minimum: • cluster: vector of cluster assignments • clusvol: ClusteredNeuroVol object • Optionally: centers, coord_centers, etc.
seed	Optional random seed for reproducibility. If NULL, uses R's current random state. Default is NULL.

Details

Many clustering algorithms assign cluster IDs sequentially as clusters are encountered during processing. When voxels are processed in memory order (which is typically z-major in 3D arrays), this creates systematic ordering where cluster ID correlates with z-position (inferior to superior in brain imaging).

This spatial ordering creates two problems:

1. **Visualization artifacts:** When viewed with continuous color scales (e.g., viridis, rainbow), ordered cluster IDs create artificial gradients that suggest spatial structure where none exists. Randomized IDs show each cluster as a distinct color patch.
2. **Interpretation bias:** Ordered IDs may mislead users into thinking cluster numbering has anatomical significance (e.g., "cluster 1 is always inferior, cluster 100 is always superior").

This function breaks the ordering by randomly permuting cluster IDs while maintaining all cluster memberships and properties.

Value

A modified cluster_result object with randomized cluster IDs. All components (cluster vector, clusvol, centers, coord_centers, etc.) are updated consistently.

Examples

```
## Not run:
# Standard workflow with ordered IDs
result <- cluster4d(vec, mask, K = 100, method = "flash3d")

# Randomize for better visualization
result_random <- randomize_cluster_ids(result, seed = 42)

# Compare visualizations
# Ordered IDs show gradient from inferior to superior
plot(result$clusvol)

# Randomized IDs show distinct color patches
plot(result_random$clusvol)

# Works with other supported clustering methods
result_snic <- cluster4d(vec, mask, n_clusters = 100, method = "snic")
result_snic_random <- randomize_cluster_ids(result_snic)

## End(Not run)
```

rena

Recursive Nearest Agglomeration (ReNA) Clustering

Description

Performs spatially-constrained hierarchical clustering using recursive 1-nearest neighbor aggregation. ReNA is a fast, linear-time agglomerative algorithm that avoids the "percolation" problem of standard hierarchical methods by using 1-NN graphs to ensure balanced cluster sizes.

Usage

```
rena(
  bvec,
  mask,
  K = 100,
  connectivity = 26,
  max_iterations = 50,
  verbose = FALSE,
  exact_k = TRUE
)
```

Arguments

bvec	A NeuroVec instance supplying the 4D data to cluster.
mask	A NeuroVol mask defining the voxels to include in clustering. Exactly finite values greater than zero are included.
K	The number of clusters to find (default 100).
connectivity	Exact masked-grid connectivity (6, 18, or 26).
max_iterations	Maximum number of recursion iterations (default 50). Algorithm stops when K clusters are reached or max_iterations is hit.
verbose	Logical; whether to print progress messages. Default FALSE.
exact_k	Logical; whether to use the shared adjacency-preserving merge and split engine to ensure exactly K clusters. A target below the number of disconnected mask components is infeasible.

Details

Algorithm Description:

ReNA recursively aggregates features using a 1-Nearest Neighbor (1-NN) graph to ensure clusters remain roughly balanced in size and spatially compact. The algorithm proceeds in iterations:

1. **Connectivity Constraint:** Clustering respects the spatial topology graph G . Distance is only calculated between spatially adjacent voxels.
2. **Similarity Calculation:** For all connected edges, compute squared Euclidean distance in feature space.

3. **1-NN Subgraph:** For every voxel, identify its single nearest neighbor. This forms a directed subgraph Q . The 1-NN graph is less likely to "percolate" (create giant components) compared to k -NN graphs where $k \geq 2$.
4. **Connected Components:** Extract weakly connected components of Q . These components become the clusters for the current iteration.
5. **Reduction:** Features within each component are averaged to form a single "super-feature" for the next iteration. The graph G is contracted so edges between new clusters exist if any constituent voxels were connected.
6. **Stopping Condition:** Repeat recursively until the number of clusters drops to the desired K . If `exact_k = TRUE`, edges are pruned by distance to ensure exactly K clusters.

Performance Characteristics:

- **Complexity:** $O(N \log N)$ per iteration where N = number of voxels
- **Memory:** $O(N)$ for graph and assignments
- **Speed:** Generally faster than iterative methods (supervoxels)
- **Iterations:** Typically 5-15 iterations to reach K clusters from N voxels

Advantages over Other Methods:

- **No percolation:** Unlike single-linkage, doesn't form one giant cluster
- **Balanced sizes:** 1-NN graph encourages roughly equal cluster sizes
- **Topology-aware:** Respects spatial structure via connectivity graph
- **Deterministic:** No random initialization
- **Fast convergence:** Linear-time operations per iteration

Parallelization Status:

Currently NOT parallelized. ReNA uses a sequential recursive algorithm that processes the graph in a hierarchical manner. Each iteration depends on the previous iteration's clustering result.

Sequential Operations::

1. **Distance Computation:** Pairwise distances for connected voxels (C++)
2. **1-NN Finding:** Each voxel finds its nearest neighbor (C++)
3. **Component Detection:** Union-Find algorithm for connected components (C++)
4. **Feature Aggregation:** Mean pooling within components (C++)
5. **Graph Contraction:** Building reduced graph for next iteration (C++)

Why Not Parallelized::

- **Sequential dependency:** Each iteration requires previous iteration's result
- **Fast per-iteration:** Linear-time operations make parallelization overhead high
- **Small iteration count:** Typically 5-15 iterations total
- **C++ optimization:** Core operations already optimized in C++

Performance Tips::

- **Reduce connectivity:** Use `connectivity=6` instead of 26 for larger graphs
- **Use smaller K :** Fewer target clusters means fewer iterations
- **Pre-smooth data:** Reduces noise, improves cluster coherence
- **Parallel alternatives:** In the unified API, `supervoxels` and `slic` support `parallel = TRUE`. The other methods do not advertise a parallel clustering contract.

Value

A list of class `rena_cluster_result` (inheriting from `cluster_result`) with the following elements:

clusvol An instance of type `ClusteredNeuroVol`.

cluster A vector of cluster indices equal to the number of voxels in the mask.

centers A matrix of cluster centers with each row representing the feature vector for a cluster.

coord_centers A matrix of spatial coordinates with each row corresponding to a cluster.

n_clusters Actual number of clusters found.

method Clustering method used ("rena").

parameters List of all parameters used.

metadata List containing iteration count and convergence information.

Note

Consider using `cluster4d` with `method = "rena"` for a standardized interface across all clustering methods.

References

Hoyos-Idrobo, A., Varoquaux, G., Kahn, J., & Thirion, B. (2019). Recursive Nearest Agglomeration (ReNA): Fast clustering for approximation of structured signals. *Pattern Recognition*, 94, 17-28.

See Also

`cluster4d` for unified interface, `supervoxels` for iterative heat kernel clustering, `snic` for non-iterative priority queue clustering

Examples

```
## Not run:
# Small synthetic example
library(neuroim2)
mask <- NeuroVol(array(1, c(10,10,10)), NeuroSpace(c(10,10,10)))
vec <- replicate(20,
  NeuroVol(array(runif(10*10*10), c(10,10,10)),
    NeuroSpace(c(10,10,10))),
  simplify=FALSE)
vec <- do.call(concat, vec)

# Run ReNA clustering
rena_res <- rena(vec, mask, K=50, connectivity=26)
print(rena_res$n_clusters)

# With verbose output
rena_res <- rena(vec, mask, K=50, verbose=TRUE)

## End(Not run)
```

rena_plus	<i>ReNA++: Edge-aware Reciprocal Multi-Level ReNA with Ward refinement</i>
-----------	--

Description

Two-stage pipeline:

1. Edge-aware reciprocal-nearest-neighbor ReNA coarsening toward $K' = \text{ceiling}(r * K)$ super-voxels.
2. Spatially-constrained Ward refinement on the super-graph to reach exactly K clusters.

Usage

```
rena_plus(
  bvec,
  mask,
  K = 100,
  r = 2,
  lambda = 1,
  grad_img = NULL,
  connectivity = 26,
  max_iterations = 50,
  verbose = FALSE,
  ...
)
```

Arguments

bvec	A NeuroVec providing voxel time-series.
mask	A NeuroVol mask; finite positive voxels are included.
K	Target number of clusters.
r	Finite over-clustering factor at least 1. Coarsening approaches $K' = \text{ceiling}(r * K)$ without applying a level that would overshoot below K' . Default 2.
lambda	Gradient penalty strength (≥ 0). 0 disables edge weighting.
grad_img	Optional numeric vector: gradient/intensity per voxel. Either length = $\text{prod}(\text{dim}(\text{mask}))$ (in which case values are subset by mask) or length = number of masked voxels. If NULL, no gradient.
connectivity	Neighborhood connectivity (6, 18, or 26). Default 26.
max_iterations	Max iterations for coarsening stage.
verbose	Logical for progress messages.
...	Reserved for future options.

Details

A reciprocal-neighbor level is retained only when applying the complete level would leave at least K' super-voxels. This prevents the coarsening stage from overshooting below K and leaves the remaining one-at-a-time adjacent merges to Ward refinement. Ward costs are recomputed from the current size-weighted means before every accepted merge; equal-cost edges are resolved by ascending super-voxel identifiers. Because the adjacency constraint can expose a new edge after a merge, individual Ward deltas need not be ordered, although every delta is non-negative and the cumulative objective is non-decreasing. When Ward refinement is needed, its complete trace and queue diagnostics are available in `result$metadata$ward`.

Value

A `cluster4d_result` with additional class `rena_plus_cluster_result`.

<code>slic4d_grad_summary</code>	<i>Visualize gradient used for seed relocation</i>
----------------------------------	--

Description

Compute and return the gradient volume that would be used for seed relocation in `slic4d_supervoxels`. Useful for debugging and visualization.

Usage

```
slic4d_grad_summary(
  bvec,
  mask,
  method = c("correlation", "intensity", "spatial")
)
```

Arguments

<code>bvec</code>	A <code>NeuroVec</code> with dims (X, Y, Z, T).
<code>mask</code>	A 3D <code>NeuroVol</code> indicating voxels to include.
<code>method</code>	One of "correlation", "intensity", "spatial".

Value

A 3D array containing the gradient values.

slic4d_supervoxels	<i>Fast 4D SLIC supervoxels (mask-aware, gradient relocation, preserve-K)</i>
--------------------	---

Description

Cluster a 4D NeuroVec (x,y,z,time) into compact, spatially contiguous 3D supervoxels using an enhanced SLIC-style algorithm with mask-aware seeding, gradient-based seed relocation, and topology-aware K preservation.

Usage

```
slic4d_supervoxels(
    bvec,
    mask,
    K,
    compactness = 10,
    max_iter = 10,
    n_threads = 0,
    step_mm = NULL,
    n_components = 0,
    feature_norm = c("zscale", "l2", "none"),
    seed_method = c("mask_poisson", "mask_grid", "grid", "farthest"),
    seed_relocate = c("none", "correlation", "intensity", "spatial"),
    seed_relocate_radius = 1L,
    connectivity = c(26L, 6L),
    strict_connectivity = TRUE,
    enforce_connectivity = TRUE,
    preserve_k = FALSE,
    topup_iters = 2L,
    min_size = 0L,
    verbose = FALSE,
    .input_contract = NULL
)
```

Arguments

bvec	A NeuroVec with dims (X, Y, Z, T). Can also be a 3D NeuroVol for structural image segmentation, which will be automatically converted to a single-timepoint NeuroVec internally.
mask	A 3D NeuroVol indicating voxels to include. Finite values greater than zero are included; zero and negative values are excluded.
K	Finite integer target number of supervoxels, between one and the number of included voxels.
compactness	Finite non-negative spatial vs feature tradeoff (like SLIC 'm'). Larger values produce more compact clusters.

<code>max_iter</code>	Positive finite integer iteration limit (default 10).
<code>n_threads</code>	Non-negative finite integer number of CPU threads. Zero requests RcppParallel's automatic thread count, one forces serial assignment, and values above one request that many threads. Assignment is intentionally serial for masks with fewer than 2,000 included voxels.
<code>step_mm</code>	Optional approximate spacing between seeds in millimeters; if NULL, computed as cubic root of bounding-box volume / K.
<code>n_components</code>	If > 0, random-project each voxel's time series to this dimension for speed (Johnson-Lindenstrauss style). 0 = use full time series.
<code>feature_norm</code>	One of "zscale", "l2", "none".
<code>seed_method</code>	One of "mask_poisson" (Poisson disk in mask), "mask_grid" (grid in mask), "grid" (regular grid), "farthest" (farthest point sampling).
<code>seed_relocate</code>	One of "correlation" (correlation gradient), "intensity" (mean intensity gradient), "spatial" (spatial gradient using adjoint), "none" (no relocation).
<code>seed_relocate_radius</code>	Search radius in voxels for gradient-based seed relocation (default 1).
<code>connectivity</code>	Neighborhood connectivity: 6 (face neighbors) or 26 (all neighbors).
<code>strict_connectivity</code>	Enforce exactly one connected component per label (default TRUE).
<code>enforce_connectivity</code>	Deprecated alias for strict connectivity. When omitted, it follows <code>strict_connectivity</code> ; if either supplied value is TRUE, connectivity is enforced.
<code>preserve_k</code>	Return exactly K non-empty labels when compatible with strict connectivity (default FALSE). See Details.
<code>topup_iters</code>	Non-negative number of native refinement iterations before the final connectivity and exact-K repair (default 2).
<code>min_size</code>	Minimum component size (voxels) to keep before relabel (default 0).
<code>verbose</code>	Logical.
<code>.input_contract</code>	Internal prevalidated input receipt used by <code>cluster4d()</code> ; direct callers should leave this as NULL.

Details

Connectivity is the final partition constraint. With `strict_connectivity = TRUE`, every returned label has exactly one connected component under the requested neighborhood. If `preserve_k = TRUE`, an adjacency-preserving repair reaches exactly K whenever K is at least the number of connected mask components. When K is below that topological minimum, strict connectivity takes precedence, the minimum feasible number of labels is returned, and a warning is emitted. Returned feature and coordinate centers are recomputed from the final public labels in the original feature space and physical coordinate system.

Value

A cluster4d_result (also inheriting from cluster_result) with elements:

- clusvol: ClusteredNeuroVol with the final labels
- cluster: integer vector of length = #masked voxels
- centers: matrix (actual_k x d_feat) center features
- coord_centers: matrix (actual_k x 3) spatial centers in mm

Note

Consider using `cluster4d` with `method = "slic"` for a standardized interface across all clustering methods.

Examples

```
## Not run:
library(neuroim2)
# Basic usage
res <- slic4d_supervoxels(bvec, mask, K = 1000, compactness = 15)

# With mask-aware seeding and gradient relocation
res <- slic4d_supervoxels(bvec, mask, K = 1000,
  seed_method = "mask_poisson",
  seed_relocate = "correlation",
  preserve_k = TRUE)

## End(Not run)
```

slice_msf

Volumetric DCT minimum-spanning-forest clustering

Description

Compresses each voxel time series into a non-DC DCT sketch, uses cosine dissimilarity on a masked 3-D neighborhood graph, and applies Felzenszwalb-Huttenlocher segmentation. Multi-run fits use seeded DCT subspace and mode-weight resampling followed by uniformly weighted consensus. Nonzero sketches are normalized to unit length. A zero-information sketch remains zero and therefore has edge dissimilarity one to every neighbor.

Usage

```
slice_msf(
  vec,
  mask,
  target_k_global = -1,
  r = 12,
  compactness = 5,
```

```

    min_size = 80,
    num_runs = 3,
    consensus = TRUE,
    stitch_z = TRUE,
    nbhd = 8,
    gamma = 0,
    k_fuse = NULL,
    min_size_fuse = NULL,
    use_features = FALSE,
    lambda = 0.7,
    z_mult = 0,
    seed = 1L,
    ensemble_fraction = 0.8
)

```

Arguments

vec	A NeuroVec or SparseNeuroVec containing the time series.
mask	A NeuroVol; exactly finite values greater than zero are included.
target_k_global	-1 for the natural segmentation or a positive exact cluster count. Exact K uses the shared adjacency-preserving split/merge engine.
r	Requested number of non-DC DCT modes. It is capped at $T - 1$.
compactness	Finite value in $[0, 10]$ controlling FH scale. Larger values use a smaller scale and generally produce finer components.
min_size	Positive minimum component size for each run.
num_runs	Positive number of runs. Values above one require consensus.
consensus	Whether to fuse a multi-run ensemble.
stitch_z	Whether graph edges may cross axial slices. When FALSE, slices are independent and a global exact-K target is unavailable.
nbhd	4, 6, or 8. Values 4 and 6 select axial neighbors; 8 selects the full diagonal neighborhood.
gamma	Reserved compatibility parameter. It must be zero; positive reliability weighting is rejected because it can collapse non-identical feature vectors to zero-distance edges.
k_fuse	Positive FH scale for consensus; defaults to the run scale.
min_size_fuse	Positive minimum consensus component size; defaults to min_size.
use_features	Include sketch similarity as well as label agreement in multi-run consensus.
lambda	Consensus mixture in $[0, 1]$; one uses label agreement only, zero uses feature similarity only. Active only when use_features = TRUE.
z_mult	DCT-sketch smoothing fraction in $[0, 1]$ across axial slices.
seed	Integer seed used to construct multi-run DCT subspaces.
ensemble_fraction	Fraction in $(0, 1]$ controlling the candidate DCT frequency pool for a multi-run ensemble.

Value

A `slice_msf_cluster_result` and `cluster4d_result`. `cluster` is in mask order, `centers` is K by T, and `coord_centers` is K by 3. Multi-run results also contain native runs. `metadata$exact_k_repair` records the natural and requested K, pre/post sizes, whether and how repair ran, connectivity, minimum size, and effective gamma, z smoothing, seed, run count, and consensus controls for both natural and exact-K calls.

Experimental status

`slice_msf()` is available for explicit evaluation but is not a recommended production method. A previous reliability-weighting path collapsed distinct feature vectors to zero-distance edges, and the former exact-K repair could create singleton-dominated partitions. Those paths are now rejected or replaced and covered by regression tests, but release-level recertification is still pending. See vignette("slice-msf") for the evaluation contract.

References

Felzenszwalb, P. F., & Huttenlocher, D. P. (2004). Efficient graph-based image segmentation. *IJCV*, 59(2), 167-181.

slice_msf_consensus	<i>Fuse Slice-MSF runs on a masked graph</i>
---------------------	--

Description

Builds neighbor-edge co-association scores from at least two run results and optionally mixes those scores with DCT-sketch similarity.

Usage

```
slice_msf_consensus(
  run_results,
  mask,
  nbhd = 8,
  k_fuse = 0.3,
  min_size_fuse = 80,
  use_features = FALSE,
  lambda = 0.7,
  stitch_z = TRUE,
  gamma = 0
)
```

Arguments

<code>run_results</code>	At least two results from <code>slice_msf_single()</code> .
<code>mask</code>	The original NeuroVol mask. Outside-mask labels are discarded.

nbhd	4, 6, or 8; see slice_msf() .
k_fuse	Positive FH scale for the consensus graph.
min_size_fuse	Positive minimum consensus component size.
use_features	Whether DCT-sketch similarity contributes to edges.
lambda	Mixture in $[0, 1]$; active when use_features = TRUE.
stitch_z	Whether consensus edges may cross axial slices.
gamma	Reserved compatibility parameter. It must be zero.

Value

A list with full-volume consensus labels and complete params.

slice_msf_single	<i>Run one Slice-MSF segmentation</i>
------------------	---------------------------------------

Description

Low-level diagnostic interface returning native full-volume labels, a signed adjacent-pair temporal-smoothness diagnostic, and an `r` used by voxel DCT sketch. Temporal smoothness does not affect segmentation.

Usage

```
slice_msf_single(
  vec,
  mask,
  r = 12,
  k = 0.32,
  min_size = 80,
  nbhd = 8,
  stitch_z = TRUE,
  gamma = 0,
  z_mult = 0,
  dct_frequencies = NULL,
  dct_weights = NULL
)
```

Arguments

vec	A NeuroVec or SparseNeuroVec containing the time series.
mask	A NeuroVol; exactly finite values greater than zero are included.
r	Requested number of non-DC DCT modes. It is capped at $T - 1$.
k	Positive FH segmentation scale.
min_size	Positive minimum component size for each run.

nbhd	4, 6, or 8. Values 4 and 6 select axial neighbors; 8 selects the full diagonal neighborhood.
stitch_z	Whether graph edges may cross axial slices. When FALSE, slices are independent and a global exact-K target is unavailable.
gamma	Reserved compatibility parameter. It must be zero; positive reliability weighting is rejected because it can collapse non-identical feature vectors to zero-distance edges.
z_mult	DCT-sketch smoothing fraction in $[0, 1]$ across axial slices.
dct_frequencies	Optional unique non-DC DCT frequencies in $[1, T - 1]$.
dct_weights	Optional positive weights paired with <code>dct_frequencies</code> .

Value

A list with full-volume labels, `temporal_smoothness`, `sketch`, and `params`.

snic	<i>SNIC: Simple Non-Iterative Clustering</i>
------	--

Description

The SNIC function performs a spatially constrained clustering on a `NeuroVec` instance using the Simple Non-Iterative Clustering (SNIC) algorithm.

Usage

```
snic(vec, mask, compactness = 5, K = 500, max_iter = 100)
```

Arguments

vec	A <code>NeuroVec</code> instance supplying the data to cluster. Can also be a 3D <code>NeuroVol</code> for structural image segmentation, which will be automatically converted to a single-timepoint <code>NeuroVec</code> internally.
mask	A <code>NeuroVol</code> mask defining the voxels to include in the clustering result. If the mask contains numeric data, finite values strictly greater than zero define the included voxels. If the mask is a <code>LogicalNeuroVol</code> , then TRUE will define the set of included voxels.
compactness	A finite numeric value in $[0, 10]$ controlling the spatial-feature mixture. Zero is feature-only, 10 is spatial-only, and larger values within the interval produce more compact clusters. Default is 5.
K	The number of clusters to find. Default is 500.
max_iter	Deprecated inactive argument retained in the function signature for compatibility. SNIC is non-iterative; supplying this argument explicitly is an error.

Details

Performance Optimization (2025):

The SNIC implementation has been **highly optimized** using lightweight C++ structs and in-place operations, providing **10x-50x speedup** over the original implementation. Key optimizations include:

- Elimination of Rcpp::List overhead in priority queue (uses lightweight struct)
- In-place centroid updates with no memory allocations
- Inline 26-connectivity neighbor iteration
- Direct pointer access for array operations
- Efficient scalar math (eliminates temporary vector allocations)

Parallelization Status:

Currently NOT parallelized. SNIC uses a sequential priority queue-based algorithm that processes voxels in order of their distance from cluster centers.

Sequential Operations::

1. **Initialization:** Gradient-based seed selection using `find_initial_points()`
 - Finds K seed points with high gradient and spatial separation
 - Sequential search through candidate voxels
2. **Priority Queue Processing** (C++ implementation):
 - Maintains a global priority queue of voxels to be assigned
 - Each voxel assignment depends on previously processed neighbors
 - Voxels are processed in order of their combined distance metric
3. **Distance Computation:** For each voxel, calculates:
 - Feature distance to nearest cluster center
 - Spatial distance weighted by compactness parameter
 - Combined into single priority score

Why Not Parallelized::

- **Sequential dependency:** Priority queue enforces strict processing order
- **Global state:** Each voxel assignment affects subsequent assignments
- **Algorithm design:** SNIC's key innovation is its non-iterative, sequential nature
- **Coherent clusters:** Sequential processing ensures connected components

Performance Characteristics::

- **Complexity:** $O(N \log N)$ where N = number of voxels
- **Memory:** $O(N)$ for priority queue and assignments
- **Speed:** Generally faster than iterative methods (supervoxels)
- **Single pass:** Processes each voxel exactly once

Performance Tips::

- **Reduce K:** Fewer clusters means less competition for voxels
- **Adjust compactness:** Higher values create more local clusters, faster processing
- **Pre-smooth data:** Reduce noise to improve gradient-based initialization
- **Use smaller masks:** Process ROIs separately if possible

- **Parallel alternative:** In the unified API, supervoxels and slic support `parallel = TRUE`. The other methods do not advertise a parallel clustering contract.

Comparison with Other Methods::

- **Faster than:** `supervoxels()` due to non-iterative nature
- **Parallel choices:** Use one of the two unified methods listed above when a tested sequential/parallel clustering contract is required.
- **More coherent than:** Methods without spatial priority (ensures connectivity)

Value

A list of class `snic_cluster_result` with the following elements:

clusvol An instance of type `ClusteredNeuroVol`.

gradvol A `NeuroVol` instance representing the spatial gradient of the reference volume.

cluster A vector of contiguous positive cluster labels, one per included mask voxel.

centers A K-by-T matrix of mean feature values in the original input space.

coord_centers A matrix of spatial coordinates with each row corresponding to a cluster.

Note

Consider using `cluster4d` with `method = "snic"` for a standardized interface across all clustering methods.

See Also

[supervoxels](#)

Examples

```
## Not run:
mask <- NeuroVol(array(1, c(20,20,20)), NeuroSpace(c(20,20,20)))
vec <- replicate(10, NeuroVol(array(runif(202020), c(20,20,20)),
NeuroSpace(c(20,20,20))), simplify=FALSE)
vec <- do.call(concat, vec)

snic_res <- snic(vec, mask, compactness=5, K=100)

## End(Not run)
```

spatial_gradient	<i>Spatial Gradient Calculation</i>
------------------	-------------------------------------

Description

The `spatial_gradient` function calculates the spatial gradient of a `NeuroVol` instance within the specified mask.

Usage

```
spatial_gradient(vol, mask, sigma = 0.5)
```

Arguments

<code>vol</code>	A <code>NeuroVol</code> instance for which the spatial gradient should be calculated.
<code>mask</code>	A <code>NeuroVol</code> mask defining the voxels to include in the spatial gradient calculation. If the mask contains numeric data, nonzero values will define the included voxels. If the mask is a LogicalNeuroVol , then <code>TRUE</code> will define the set of included voxels.
<code>sigma</code>	A numeric value controlling the spatial weighting function. Default is 0.5.

Value

A `NeuroVol` instance containing the spatial gradient values for the input `vol`.

See Also

[spatial_laplacian](#), [weighted_spatial_adjacency](#)

Examples

```
## Not run:
mask <- neuroim2::NeuroVol(array(1, c(20,20,20)), neuroim2::NeuroSpace(c(20,20,20)))
input_vol <- neuroim2::NeuroVol(array(runif(20*20*20), c(20,20,20)),
  neuroim2::NeuroSpace(c(20,20,20)))
gradient_vol <- spatial_gradient(input_vol, mask)

## End(Not run)
```

suggest_cluster4d_params

Suggest cluster4d parameters based on data characteristics

Description

Provides parameter recommendations based on data size and user priorities. Experimental methods are reported separately and are never returned as the recommended method.

Usage

```
suggest_cluster4d_params(
  n_voxels,
  n_timepoints,
  priority = c("balanced", "speed", "quality", "memory")
)
```

Arguments

n_voxels	Number of voxels in mask
n_timepoints	Number of time points
priority	What to optimize for: "speed", "quality", "memory", or "balanced"

Value

A list with suggested parameters for supported methods plus `experimental_methods` and an `experimental_note`. Experimental methods are not recommended or assigned suggested parameter sets.

Examples

```
# Get parameter suggestions for a typical fMRI dataset
params <- suggest_cluster4d_params(50000, 200, priority = "balanced")
print(params$recommended_method)
print(params$n_clusters)

# Speed-optimized parameters for large dataset
speed_params <- suggest_cluster4d_params(100000, 300, priority = "speed")
print(speed_params$recommended_method)

# Quality-optimized parameters for smaller dataset
quality_params <- suggest_cluster4d_params(10000, 150, priority = "quality")
print(quality_params$n_clusters)

# Memory-efficient parameters
memory_params <- suggest_cluster4d_params(200000, 100, priority = "memory")
print(memory_params$recommended_method)
```

```
summary.cluster4d_result
```

Summarize cluster4d result

Description

Summarize cluster4d result

Usage

```
## S3 method for class 'cluster4d_result'
summary(object, ...)
```

Arguments

object	A cluster4d_result object
...	Additional arguments (ignored)

Value

A summary list (invisibly)

```
supervoxel_cluster_surface
```

Supervoxel Clustering on a Surface

Description

Cluster feature data on a cortical surface or mesh using a supervoxel-like approach.

Usage

```
supervoxel_cluster_surface(
  bsurf,
  K = 500,
  sigma1 = 1,
  sigma2 = 5,
  iterations = 50,
  connectivity = 6,
  use_medoid = FALSE
)
```

Arguments

<code>bsurf</code>	A NeuroSurface or similar object with geometry, coords, and data.
<code>K</code>	Number of clusters.
<code>sigma1</code>	Heat kernel bandwidth for feature similarity (data vectors).
<code>sigma2</code>	Heat kernel bandwidth for spatial similarity (coordinate vectors).
<code>iterations</code>	Max iterations.
<code>connectivity</code>	Neighborhood size on the surface (e.g., # of nearest mesh neighbors).
<code>use_medoid</code>	Whether to use medoids for cluster centers.

Value

A list with:

- clusvol** A NeuroSurface storing the final clustering result.
- clusters** Integer vector of cluster assignments (one per vertex).
- centers** Matrix of cluster centers.
- coord_centers** Matrix of spatial centroid coordinates.
- index_sets** List of vertex indices for each cluster.

supervoxel_cluster_time

Supervoxel Clustering in Time

Description

Cluster feature matrix (rows = time points) in a "supervoxel" style but over temporal dimension.

Usage

```
supervoxel_cluster_time(
  feature_mat,
  K = min(nrow(feature_mat), 100),
  sigma1 = 1,
  sigma2 = 3,
  iterations = 50,
  TR = 2,
  filter = list(lp = 0, hp = 0),
  use_medoid = FALSE,
  nreps = 5
)
```

Arguments

<code>feature_mat</code>	A matrix (nrows = time points, ncols = features) or vice versa.
<code>K</code>	Number of clusters.
<code>sigma1</code>	Heat kernel bandwidth for feature similarity (data vectors).
<code>sigma2</code>	Heat kernel bandwidth for spatial similarity (coordinate vectors).
<code>iterations</code>	Maximum number of cluster iterations.
<code>TR</code>	Repetition time (seconds).
<code>filter</code>	List specifying optional frequency filters, e.g., <code>list(lp=0.1, hp=0)</code> .
<code>use_medoid</code>	Whether to use medoids for cluster centers.
<code>nreps</code>	Number of repeated initializations.

Value

A list of cluster results (one per repetition), each of which has the same structure as `supervoxel_cluster_fit()`.

Examples

```
## Not run:
feature_mat <- matrix(rnorm(100 * 10), 100, 10)
library(future)
plan(multicore)
cres <- supervoxel_cluster_time(t(feature_mat), K=5)

## End(Not run)
```

supervoxels	<i>Supervoxel Clustering (3D volumes)</i>
-------------	---

Description

Cluster a NeuroVec instance into a set of spatially constrained clusters.

Usage

```
supervoxels(
  bvec,
  mask,
  K = 500,
  sigma1 = NULL,
  sigma2 = 2.5,
  iterations = 50,
  connectivity = 27,
  use_medoid = FALSE,
  use_gradient = TRUE,
  alpha = 0.5,
```

```

parallel = TRUE,
grain_size = 100,
num_threads = NULL,
verbose = FALSE,
converge_thresh = 0.001
)

```

Arguments

bvec	A NeuroVec instance supplying the data to cluster. Can also be a 3D NeuroVol for structural image segmentation, which will be automatically converted to a single-timepoint NeuroVec internally.
mask	A NeuroVol mask defining the voxels to include. If numeric, nonzero = included.
K	The number of clusters to find (default 500).
sigma1	The bandwidth of the heat kernel for the data vectors.
sigma2	The bandwidth of the heat kernel for the coordinate vectors.
iterations	The maximum number of cluster iterations.
connectivity	The number of nearest neighbors defining the neighborhood.
use_medoid	Logical; whether to use medoids rather than means for cluster centers.
use_gradient	Logical; use the image gradient to initialize clusters if possible.
alpha	The relative weighting of data similarity vs spatial similarity; alpha=1 = all data weighting, alpha=0 = purely spatial weighting.
parallel	Logical; whether to use parallel processing for cluster assignment updates. Default is TRUE. Parallel processing is automatically disabled for small datasets (<1000 voxels).
grain_size	Integer; the minimum number of voxels to process per parallel task. Default is 100. Smaller values provide better load balancing but increase overhead.
num_threads	Optional integer to override the number of threads used by RcppParallel (defaults to package/global setting). Ignored if parallel = FALSE.
verbose	Logical; whether to print detailed progress messages including convergence metrics. Default is FALSE.
converge_thresh	Numeric; convergence threshold as proportion of voxels switching clusters. Algorithm stops when switch ratio falls below this value. Default is 0.001 (0.1% of voxels).

Details

The algorithm:

1. Scale input data (bvec) so each feature dimension is centered and scaled.
2. If use_gradient = TRUE, initialize cluster seeds using gradient-based heuristics.

3. Run an iterative, spatially-constrained clustering that updates voxel assignments based on both feature similarity (bandwidth σ_1) and spatial proximity (bandwidth σ_2), weighted by α .
4. Return the final clusters, plus the feature-space and coordinate-space centers.

Exact-K and parallel contracts:

Every assignment center must have a positive membership count. If an update empties one or more labels, distinct voxels are deterministically moved from non-singleton clusters in descending combined feature/spatial loss order. Empty all-zero centers are therefore never passed to an assignment kernel. Since $K \leq N$ is required, this reseed policy is feasible and the returned labels, center rows, and requested K agree on every convergence path.

With `parallel = TRUE`, voxel assignment uses `RcppParallel` when there are more than 1000 included voxels; smaller inputs use the same sequential scoring kernel. Parallel and sequential kernels implement the same scoring and deterministic tie rule. Centroid reduction may also run in parallel for sufficiently large K . `metadata$algorithm$parallel_used` reports whether the parallel assignment path actually ran. No architecture-specific fallback is applied.

Value

A list (of class `cluster_result`) with elements:

<code>clusvol</code>	ClusteredNeuroVol containing the final clustering.
<code>cluster</code>	Integer vector of cluster assignments for each voxel.
<code>centers</code>	K -by- T matrix of final cluster means in the original feature space.
<code>coord_centers</code>	K -by-3 matrix of final spatial means in physical units.
<code>metadata</code>	Algorithm diagnostics, including whether parallel assignment was requested and used and any deterministic empty-label reseeds.

Note

Consider using `cluster4d` with `method = "supervoxels"` for a standardized interface across all clustering methods.

Examples

```
## Not run:
mask <- NeuroVol(array(1, c(20,20,20)), NeuroSpace(c(20,20,20)))
bvec <- replicate(10,
  NeuroVol(array(runif(20*20*20), c(20,20,20)),
    NeuroSpace(c(20,20,20))),
  simplify=FALSE)
bvec <- do.call(concat, bvec)
cres1 <- supervoxels(bvec, mask, K=100, sigma1=1, sigma2=3)

## End(Not run)
```

supervoxels_flash3d *FLASH-3D: Fast 3D Superclustering for fMRI*

Description

Performs spatially-constrained clustering using Fast Low-rank Approximate Superclusters for Hemodynamics (FLASH-3D). This algorithm uses DCT-based temporal hashing and 3D jump-flood propagation for efficient clustering.

Usage

```
supervoxels_flash3d(
    vec,
    mask,
    K,
    lambda_s = 0.6,
    lambda_t = 1,
    lambda_g = 0,
    rounds = 2L,
    bits = 64L,
    dctM = 12L,
    vox_scale = NULL,
    barrier = NULL,
    verbose = FALSE
)
```

Arguments

vec	A NeuroVec instance supplying the 4D data to cluster
mask	A NeuroVol mask defining the voxels to include in the clustering result. If the mask contains numeric data, finite values strictly greater than zero define the included voxels. If the mask is a LogicalNeuroVol , then TRUE will define the set of included voxels.
K	The number of clusters to find
lambda_s	Finite non-negative target spatial weight (default 0.6). Multi-round optimization warms up from half this value and finishes at the requested value.
lambda_t	Finite non-negative temporal weight for Hamming distance of time-series hashes (default 1.0).
lambda_g	Finite non-negative boundary-field weight (default 0.0). If positive, barrier is required.
rounds	Number of outer rounds (seed→flood→recenter). Default 2, typically 2-3 is sufficient.
bits	Maximum temporal hash length (64 or 128). The effective number of non-redundant comparisons is at most choose(dctM, 2).
dctM	Number of low DCT coefficients to rank-hash (default 12, range 4-32)

vox_scale	Voxel size scaling for spatial distance, e.g., c(dx,dy,dz). Default c(1,1,1)
barrier	Optional non-negative finite 3D boundary field with the same geometry as the mask. A candidate site is penalized when its seed and the target voxel have different boundary-field values.
verbose	Logical indicating whether to print progress messages

Details

FLASH-3D uses a novel approach combining:

- DCT-based temporal feature hashing for fast similarity computation
- Jump-flood algorithm for efficient spatial propagation
- Blue-noise seeding for optimal initial cluster placement
- Annealing of spatial weights to encourage compact clusters

The algorithm is particularly efficient for large-scale fMRI data, offering significant speed improvements over iterative methods while maintaining clustering quality.

Value

A list of class `cluster_result` with the following elements:

clusvol An instance of type `ClusteredNeuroVol`

cluster An integer vector of cluster assignments for each voxel in mask

centers A matrix of cluster centers in feature space ($K \times T$)

coord_centers A matrix of spatial coordinates of cluster centers ($K \times 3$)

K The number of clusters

method A character string indicating the method used ("FLASH-3D")

Note

Consider using `cluster4d` with `method = "flash3d"` for a standardized interface across all clustering methods.

References

FLASH-3D algorithm for fast superclustering of fMRI data (2025)

See Also

`supervoxels`, `snic`, `slic4d_supervoxels`

Examples

```
## Not run:
# Basic usage
result <- supervoxels_flash3d(vec, mask, K = 100)

# With custom parameters
result <- supervoxels_flash3d(vec, mask, K = 100,
                              lambda_s = 0.8, lambda_t = 1.2,
                              bits = 128, dctM = 16)

# With barrier for anatomy-aware clustering
barrier_vol <- create_anatomical_barrier(mask)
result <- supervoxels_flash3d(vec, mask, K = 100,
                              lambda_g = 0.5, barrier = barrier_vol)

## End(Not run)
```

synthetic_time_patterns

Generate synthetic temporal patterns for cluster signals

Description

Creates distinct temporal patterns for each cluster. Used internally by [generate_synthetic_volume](#) but exported for custom synthetic data generation.

Usage

```
synthetic_time_patterns(
  n_clusters,
  n_time,
  pattern_types = c("orthogonal", "harmonic", "random")
)
```

Arguments

n_clusters	Number of clusters/patterns to generate.
n_time	Number of time points.
pattern_types	Type of patterns: "orthogonal" (QR-decomposition for maximally distinct signals), "harmonic" (phase/frequency-shifted sinusoids), or "random" (random linear combinations of harmonics).

Value

A matrix of dimension n_clusters x n_time with one temporal pattern per row.

tessellate

Tessellate a Mask Volume into K Clusters using K-means

Description

This function tessellates a given mask volume into K clusters using k-means clustering applied to spatial coordinates. It returns a clustered mask volume object.

Usage

```
tessellate(mask, K = 100)
```

Arguments

mask	A NeuroVol object representing the mask volume.
K	An integer value specifying the number of clusters (default: 100). If K exceeds the number of nonzero voxels, a warning is issued and K is set to the number of nonzero voxels.

Value

An instance of ClusteredNeuroVol representing the clustered mask volume.

Examples

```
## Not run:
# Assuming you have a NeuroVol object 'mask' and you want to create 150 clusters
clustered_volume <- tessellate(mask, K = 150)

## End(Not run)
```

test_all_clustering_methods

Test All Clustering Methods on Synthetic Data

Description

Comprehensive test of all available clustering methods on synthetic ground truth data with cubic clusters.

Usage

```
test_all_clustering_methods(
  noise_levels = c(0, 0.1, 0.5),
  methods = c("flash3d", "corr_slic", "snic", "slic", "slice_msf", "supervoxels", "g3s",
    "rena"),
  n_clusters = 27,
  seed = 42
)
```

Arguments

noise_levels	Vector of noise standard deviations to test. Default c(0, 0.1, 0.5) tests perfect, low noise, and moderate noise.
methods	Vector of method names to test. Default tests all main methods.
n_clusters	Number of ground truth clusters (must be perfect cube). Default 27.
seed	Random seed for reproducibility.

Value

A data.frame with columns: method, noise_sd, ari, nmi, accuracy, n_clusters, time, perfect

Examples

```
## Not run:
# Test all methods with 27 clusters
results <- test_all_clustering_methods()
print(results)

# Test with 8 clusters (simpler)
results <- test_all_clustering_methods(n_clusters = 8)

# Test specific methods
results <- test_all_clustering_methods(
  methods = c("flash3d", "corr_slic", "snic"),
  noise_levels = c(0, 0.2)
)

## End(Not run)
```

test_clustering_method

Test Clustering Method on Synthetic Data

Description

Runs a clustering method on synthetic ground truth data and reports accuracy metrics.

Usage

```
test_clustering_method(
  method,
  noise_sd = 0,
  n_clusters = 27,
  seed = 42,
  verbose = TRUE,
  ...
)
```

Arguments

method	Clustering method name (e.g., "supervoxels", "snic", "flash3d")
noise_sd	Noise level for synthetic data. Default 0 (perfect separability).
n_clusters	Number of ground truth clusters (must be perfect cube). Default 27.
seed	Random seed for reproducibility.
verbose	Print results. Default TRUE.
...	Additional arguments passed to cluster4d()

Value

A list with:

metrics	Output from clustering_accuracy()
result	Full clustering result object
data	Synthetic data used
time	Elapsed time in seconds

Examples

```
## Not run:
# Test FLASH-3D with 27 cubic clusters
test_clustering_method("flash3d")

# Test with 8 clusters
test_clustering_method("slic", n_clusters = 8)

# Test with noise
test_clustering_method("snic", noise_sd = 0.5)

## End(Not run)
```

transform_new_data_svd

Transform New Data Using Existing SVD Compression

Description

Applies a previously computed SVD compression to new data, ensuring consistency across multiple runs or when adding new voxels.

Usage

```
transform_new_data_svd(new_data, compression_result)
```

Arguments

new_data Numeric matrix ($N_{\text{new}} \times T$) with same number of columns as the original training data.

compression_result List returned by [compress_features_svd](#) containing the rotation matrix and scaling parameters.

Value

Compressed and normalized feature matrix ($N_{\text{new}} \times M$) where M is the number of components in the original compression.

Examples

```
# Train on subset of data
train_data <- matrix(rnorm(500 * 300), 500, 300)
compression <- compress_features_svd(train_data, n_components = 15)

# Apply to new data
test_data <- matrix(rnorm(100 * 300), 100, 300)
compressed_test <- transform_new_data_svd(test_data, compression)
```

validate_cluster4d

Validate cluster4d result

Description

Checks validity and quality of clustering results.

Usage

```
validate_cluster4d(result, vec = NULL, mask = NULL, tolerance = 1e-08)
```

Arguments

result	A cluster4d_result object
vec	Original NeuroVec data, required to verify feature centers.
mask	Original mask, required to verify geometry, voxel order, and physical coordinate centers.
tolerance	Relative numeric tolerance used when independently recomputing centers and coordinate centers.

Value

A list with valid, errors, warnings, and a compact summary. Any schema or consistency defect sets valid = FALSE.

Index

acsc, [3](#)
as.cl_hard_partition, [57](#)

check_cluster_ordering, [5](#)
cl_class_ids, [6, 7](#)
cl_class_ids.cluster_result, [6](#)
cl_consensus, [57](#)
cl_ensemble, [57](#)
cluster4d, [8, 69, 74, 80, 87, 89](#)
cluster4d_acsc, [11, 12](#)
cluster4d_brsslic, [13](#)
cluster4d_commute, [11, 16](#)
cluster4d_corrslic, [17](#)
cluster4d_flash3d, [11, 20](#)
cluster4d_g3s, [11, 21, 38, 49](#)
cluster4d_mcl, [11, 22](#)
cluster4d_rena, [11, 24](#)
cluster4d_rena_plus, [11, 25](#)
cluster4d_slic, [11, 26](#)
cluster4d_slice_msf, [11, 28](#)
cluster4d_snic, [11, 30](#)
cluster4d_supervoxels, [11, 31](#)
cluster_metrics, [7](#)
ClusteredNeuroVol, [56, 57, 60, 69, 80, 89](#)
clustering_accuracy, [33](#)
commute_cluster, [17, 34](#)
compare_cluster4d, [35](#)
compress_features_svd, [36, 47, 94](#)
compute_centroids, [39](#)
compute_functional_gradient, [40, 49](#)
cutree, [60](#)

detrend_basis_cpp, [41](#)
detrend_dct_cpp, [42](#)
detrend_poly_cpp, [43](#)
detrend_time, [43, 61](#)
detrend_time_cpp, [45](#)

er_ml_rena, [45](#)
estimate_optimal_components, [46](#)

find_gradient_seeds, [50](#)
find_gradient_seeds_g3s, [47](#)
find_initial_points, [49](#)

generate_synthetic_volume, [50, 90](#)

hclust, [60](#)

initialize_clusters, [50](#)
is.cl_partition, [51, 52](#)
is.cl_partition.cluster_result, [51](#)

knn_shrink, [52](#)

LogicalNeuroVol, [78, 81, 88](#)

make_block_synthetic, [53](#)
make_dct_basis, [53](#)
make_poly_basis, [54](#)
make_synthetic_clusters, [55](#)
merge_clus, [56](#)
merge_clus.cluster_result, [57](#)
meta_clust, [58](#)
meta_clust.cluster_result, [58](#)

neurocluster_simd_info, [60](#)
NeuroVec, [52, 86](#)
NeuroVol, [52, 72, 78, 86](#)
normalize_detrend, [61](#)
normalize_detrend_cpp, [62](#)
normalize_volumes, [61, 62](#)
normalize_volumes_cpp, [63](#)

plot.cluster4d_result, [64](#)
print.cluster4d_result, [64](#)
print.g3s_result, [65](#)

randomize_cluster_ids, [65](#)
rena, [67](#)
rena_plus, [45, 70](#)

slic4d_grad_summary, [71](#)

slic4d_supervoxels, [11](#), [72](#), [89](#)
slice_msf, [11](#), [74](#)
slice_msf(), [29](#), [77](#)
slice_msf_consensus, [76](#)
slice_msf_single, [77](#)
slice_msf_single(), [76](#)
snic, [11](#), [69](#), [78](#), [89](#)
spatial_gradient, [49](#), [81](#)
spatial_laplacian, [81](#)
suggest_cluster4d_params, [82](#)
suggest_cluster4d_params(), [28](#)
summary.cluster4d_result, [83](#)
supervoxel_cluster_surface, [83](#)
supervoxel_cluster_time, [84](#)
supervoxels, [11](#), [69](#), [80](#), [85](#), [89](#)
supervoxels_flash3d, [11](#), [88](#)
synthetic_time_patterns, [90](#)

tessellate, [91](#)
test_all_clustering_methods, [91](#)
test_clustering_method, [92](#)
transform_new_data_svd, [38](#), [94](#)

validate_cluster4d, [94](#)

weighted_spatial_adjacency, [81](#)