Package: fmrilatent (via r-universe)

June 3, 2026

Type Package

Title Latent Space Representations of fMRI Data

Version 0.1.0

Maintainer Bradley Buchsbaum <brad.buchsbaum@gmail.com>

Description Provides S4 classes and methods for representing neuroimaging data in latent spaces. The LatentNeuroVec class offers a memory-efficient representation using matrix factorization (basis x loadings plus a per-voxel offset), enabling compact storage and efficient computation for PCA, temporal bases, graph and wavelet dictionaries, Slepian bases, HRBF atoms, and related fMRI decompositions. Integrates with the neuroim2 package for standard neuroimaging data structures.

License GPL (>= 3)

URL https://github.com/bbuchsbaum/fmrilatent

BugReports https://github.com/bbuchsbaum/fmrilatent/issues

Encoding UTF-8

LazyData false

RoxygenNote 7.3.3

Imports cli, crayon, graphics, Matrix, methods, digest, neuroim2, rlang, stats, utils, Rcpp

Suggests albersdown, knitr, rmarkdown, testthat (>= 3.0.0), RSpectra, neuroatlas, neurosurf, rgsp, bench, ggplot2

VignetteBuilder knitr

Config/testthat/edition 3

LinkingTo Rcpp, RcppEigen

SystemRequirements OpenMP (optional)

Collate 'RcppExports.R' 'latent_utils.R' 'latent_handles.R' 'all_class.R' 'all_generic.R' 'encode.R' 'reduction.R' 'heat_wavelet.R' 'awpt.R' 'benchmark_roundtrip.R' 'bspline_basis.R' 'codec_boldzip.R' 'codec_boldzip_decompose.R'

'codec_boldzip_diagnostics.R' 'codec_boldzip_quantize.R'
'codec_boldzip_spatial.R' 'codec_boldzip_validate.R'
'compat_profile.R' 'encoder_utilities.R' 'cpp_boundary.R'
'dct_basis.R' 'diffusion_wavelet.R'
'diffusion_wavelet_handle.R' 'encode_methods_hierarchical.R'
'encode_methods_space.R' 'encode_methods_st.R'
'encode_methods_time.R' 'encode_operator.R'
'encoder_validators.R' 'encode_spec.R' 'encode_template.R'
'encode_transport_solve.R' 'encoder_registry.R'
'fmrilatent-package.R' 'graph_bridge.R' 'haar_wavelet.R'
'heat_wavelet_handle.R' 'hierarchical_helpers.R'
'hierarchical_template.R' 'hrbf.R' 'implicit_latent.R'
'latent_shared_validation.R' 'slepian_handles.R'
'latent_neurovec_materialize.R' 'latent_surface_vector.R'
'latent_bilateral_surface_vector.R' 'transport_latent.R'
'latent_block_vector.R' 'latent_dct_heatwavelet.R'
'latent_neurovector.R' 'latent_indexing.R' 'latent_methods.R'
'parcel_basis.R' 'pca_spatial.R' 'searchlight_utils.R'
'shared_structure.R' 'slepian_plot.R' 'slepian_spatial.R'
'slepian_spatiotemporal.R' 'slepian_temporal.R'
'spatial_plot.R' 'surface_template.R' 'wavelet_active.R'
'zzz.R'

Config/Needs/website albersdown

Config/pak/sysreqs make libicu-dev

Repository https://bbuchsbaum.r-universe.dev

Date/Publication 2026-06-03 14:11:20 UTC

RemoteUrl https://github.com/bbuchsbaum/fmrilatent

RemoteRef HEAD

RemoteSha 54efa9000a79fdb92bafb06b52680da9aa589f3f

Contents

`[[,LatentNeuroVec,numeric-method`

Extract Elements from LatentNeuroVec

Description

Extract elements from a `LatentNeuroVec`. Use `[[` to extract a single volume as a `SparseNeuroVol`, or `[` for 4D subsetting.

Usage

```
## S4 method for signature 'LatentNeuroVec,numeric'
x[[i]]

## S4 method for signature 'LatentNeuroVec,numeric,numeric,ANY'
x[i, j, k, l, ..., drop = TRUE]

## S4 method for signature 'LatentNeuroVec,matrix,missing,ANY'
x[i, j, k, l, ..., drop = TRUE]

## S4 method for signature 'LatentNeuroVec,ANY,ANY,ANY'
x[i, j, k, l, ..., drop = TRUE]
```

Arguments

`x`	A `LatentNeuroVec-class` object.
`i`	Numeric index for first dimension (x-axis) or volume index for `[[`.
`j`	Numeric index for second dimension (y-axis).
`k`	Numeric index for third dimension (z-axis).
`l`	Numeric index for fourth dimension (time).
`...`	Additional arguments (unused).
`drop`	Logical, whether to drop dimensions (default TRUE).

Value

For `[[`: A `SparseNeuroVol-class` containing the computed volume. For `[`: An array of extracted values.

Examples

```
mask <- neuroim2::LogicalNeuroVol(
  array(TRUE, dim = c(2, 2, 1)),
  neuroim2::NeuroSpace(c(2, 2, 1))
)
lvec <- LatentNeuroVec(
  basis = matrix(1:6, nrow = 3),
```

```
  loadings = matrix(seq_len(8) / 10, nrow = 4),
  space = neuroim2::NeuroSpace(c(2, 2, 1, 3)),
  mask = mask,
  expect_dense = TRUE
)
# Extract volumes
vol1 <- lvec[[1]]
vol_mid <- lvec[[2]]
vol_last <- lvec[[dim(lvec)[4]]]
```

`analysis_transform` *Describe the transform from raw to analysis coordinates*

Description

Describe the transform from raw to analysis coordinates

Usage

```
analysis_transform(x, ...)

## S4 method for signature 'LatentNeuroSurfaceVector'
analysis_transform(x, ...)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
analysis_transform(x, ...)

## S4 method for signature 'ImplicitLatent'
analysis_transform(x, ...)

## S4 method for signature 'LatentNeuroVec'
analysis_transform(x, ...)

## S4 method for signature 'BlockLatentNeuroVector'
analysis_transform(x, ...)
```

Arguments

`x`	A latent object.
`...`	Additional arguments passed to methods.

Value

Transform descriptor or `NULL`. Downstream model-fitting code should ordinarily consume `coef_time(x, "analysis")` rather than reasoning about raw coordinates directly.

`as.array,LatentNeuroVec-method`
Reconstruct LatentNeuroVec as a 4D array

Description

Reconstruct LatentNeuroVec as a 4D array

Usage

```
## S4 method for signature 'LatentNeuroVec'
as.array(x, ...)
```

Arguments

`x`	A `LatentNeuroVec` object.
`...`	Ignored.

Value

A 4D numeric array with dimensions (x, y, z, time).

`as.array.ImplicitLatent`
Reconstruct an ImplicitLatent as an array

Description

Reconstruct an ImplicitLatent as an array

Usage

```
## S3 method for class 'ImplicitLatent'
as.array(x, time_idx = NULL, roi_mask = NULL, ...)
```

Arguments

`x`	An `ImplicitLatent` object.
`time_idx`	Optional integer time indices to keep.
`roi_mask`	Optional logical ROI mask; voxels outside the ROI are zero.
`...`	Additional arguments passed to the decoder.

Value

A numeric array with dimensions `c(x, y, z, time)`.

`as.matrix,LatentNeuroSurfaceVector-method`
Reconstruct LatentNeuroVec as a matrix (time x voxels)

Description

Reconstruct LatentNeuroVec as a matrix (time x voxels)

Usage

```
## S4 method for signature 'LatentNeuroSurfaceVector'
as.matrix(x, ...)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
as.matrix(x, ...)

## S4 method for signature 'BlockLatentNeuroVector'
as.matrix(x, ...)

## S4 method for signature 'LatentNeuroVec'
as.matrix(x, ...)
```

Arguments

`x`	A `LatentNeuroVec` object.
`...`	Ignored.

Value

A numeric matrix with rows = time points, columns = mask voxels.

`as.matrix.GroupDeltaLoadings`
Coerce group-plus-delta loadings to a dense matrix

Description

Coerce group-plus-delta loadings to a dense matrix

Usage

```
## S3 method for class 'GroupDeltaLoadings'
as.matrix(x, ...)
```

Arguments

x	A 'GroupDeltaLoadings' object.
...	Ignored.

Value

Dense matrix of materialized loadings.

---

as.matrix.HaarLatent *Convert HaarLatent to matrix*

Description

Convert HaarLatent to matrix

Usage

```
## S3 method for class 'HaarLatent'
as.matrix(x, ...)
```

Arguments

x	HaarLatent object
...	Additional arguments (unused)

Value

Matrix of reconstructed values

---

as.matrix.ImplicitLatent *Reconstruct an ImplicitLatent as a matrix*

Description

Reconstruct an ImplicitLatent as a matrix

Usage

```
## S3 method for class 'ImplicitLatent'
as.matrix(x, time_idx = NULL, roi_mask = NULL, ...)
```

Arguments

`x`	An `ImplicitLatent` object.
`time_idx`	Optional integer time indices to keep.
`roi_mask`	Optional logical ROI mask for spatial subsetting.
`...`	Additional arguments passed to the decoder.

Value

A numeric matrix with rows = time and columns = voxels within the requested mask support.

`as_boldzip_spatial_basis` *Coerce a spatial object to a BOLDZip-SR spatial basis*

Description

This helper lets the standalone BOLDZip-SR codec consume matrix-like shared basis assets without registering BOLDZip as an ‘encode()‘ family. Matrix and template inputs are used as the detail basis by default and are orthonormalized because BOLDZip projection currently uses the transpose as the analysis operator.

Usage

```
as_boldzip_spatial_basis(x, ...)
```

Arguments

`x`	A ‘BoldZipSRSpatialBasis‘, matrix-like object, shared reference, or template object supporting [template_loadings()].
`...`	Additional arguments reserved for methods. The default method accepts ‘label‘, ‘role‘, and ‘orthonormalize‘.

Value

A ‘BoldZipSRSpatialBasis‘ object.

as_cluster_reduction *Convert a ClusteredNeuroVol to a ClusterReduction*

Description

Bridges the `neuroim2::ClusteredNeuroVol` parcellation representation to fmrilatent's `ClusterReduction` class, preserving label metadata.

Usage

```
as_cluster_reduction(cvol)
```

Arguments

`cvol`	A `ClusteredNeuroVol` object (from **neuroim2**).

Value

A `ClusterReduction` object with label metadata in `info`.

as_haar_latent *Convert to HaarLatent class*

Description

Convert to HaarLatent class

Usage

```
as_haar_latent(obj)
```

Arguments

`obj`	Object to convert

Value

Object with HaarLatent class added

as_hrbf_latent *Attach HRBF metadata to an existing LatentNeuroVec*

Description

Attach HRBF metadata to an existing LatentNeuroVec

Usage

```
as_hrbf_latent(lvec, params, centres = NULL, sigmas = NULL)
```

Arguments

lvec	A LatentNeuroVec object
params	HRBF parameters list
centres	Optional matrix of HRBF centres
sigmas	Optional vector of HRBF sigmas

Value

The LatentNeuroVec with HRBF metadata attached

as_implicit_latent *Coerce an object to an ImplicitLatent decoder*

Description

Coerce an object to an ImplicitLatent decoder

Usage

```
as_implicit_latent(x, ...)
```

Arguments

x	Object to coerce.
...	Additional arguments passed to methods.

Value

An object of class 'ImplicitLatent'.

as_implicit_latent.BoldZipSR

Coerce a BOLDZip-SR payload to an ImplicitLatent

Description

Coerce a BOLDZip-SR payload to an ImplicitLatent

Usage

```
## S3 method for class 'BoldZipSR'
as_implicit_latent(x, mask = NULL, domain = NULL, support = NULL, ...)
```

Arguments

`x`	A 'BoldZipSR' object.
`mask`	Optional logical 3D array or 'LogicalNeuroVol' for volumetric wrapping.
`domain`	Optional decoded output domain.
`support`	Optional decoded support. Defaults to row indices.
`...`	Additional arguments ignored.

Value

An 'ImplicitLatent' whose decoder returns matrices as 'time x voxels', matching the rest of the implicit latent API.

as_portable_linear_map

Coerce an operator to the portable linear-map contract

Description

Normalizes an incoming object to the canonical `portable_linear_map` representation used internally by `encode_operator()`, `decoder()`, and related routines. Matrices and `Matrix::Matrix` objects are wrapped with closure-based `forward`/`adjoint_apply` callbacks; list inputs are validated and re-normalized with top-level `source_support` / `target_support` preferred over `provenance$*` fallbacks.

Usage

```
as_portable_linear_map(
  x,
  source_domain_id = "",
  target_domain_id = "",
  source_support = NULL,
  target_support = NULL,
  provenance = list(),
  context = "portable linear map"
)
```

Arguments

`x`	An input operator. Either a base `matrix`, `Matrix::Matrix`, or a list satisfying the `portable_linear_map` contract.
`source_domain_id`, `target_domain_id`	Optional character tags applied only when `x` is a matrix or `Matrix::Matrix`. Ignored for list inputs (which carry their own).
`source_support`, `target_support`	Optional support descriptors applied only when `x` is a matrix or `Matrix::Matrix`. Typical values are a `LogicalNeuroVol`, a 3D logical mask, integer vertex indices, or `NULL`. Ignored for list inputs.
`provenance`	Optional provenance list applied only when `x` is a matrix or `Matrix::Matrix`. Ignored for list inputs.
`context`	Optional label used in error messages.

Value

A list in the canonical portable-linear-map form.

See Also

`portable_linear_map`, `validate_portable_linear_map`.

Examples

```
m <- matrix(1:6, nrow = 2)
op <- as_portable_linear_map(m, target_domain_id = "native")
op$forward(c(1, 1, 1))
```

`awpt_basis_template` *Build an AWPT basis template*

Description

Build an AWPT basis template

Usage

```
awpt_basis_template(
  parcellation,
  basis_spec = basis_awpt_wavelet(),
  loadings = NULL,
  anatomical_operator = NULL,
  conductance = NULL,
  coefficient_roughness = NULL,
  center = FALSE,
  ridge = 1e-08,
  label = "awpt_wavelet",
  ...
)
```

Arguments

`parcellation`	A `ClusterReduction` or `ClusteredNeuroVol`.
`basis_spec`	An AWPT basis specification created by `basis_awpt_wavelet()`.
`loadings`	Optional explicit template loadings matrix. When supplied, fmrilatent skips wave-packet lifting and uses these loadings directly as the decoder basis B.
`anatomical_operator`	Optional field-space roughness operator on the template domain. When supplied, the coefficient roughness is computed as $Q = B^T L B$. In the current v1 implementation this affects the roughness penalty, not the basis construction itself.
`conductance`	Optional symmetric field-space conductance matrix. When supplied, fmrilatent builds the corresponding graph Laplacian and then forms $Q = B^T L B$. As with `anatomical_operator`, this shapes the v1 roughness model rather than directly adapting the lifted basis.
`coefficient_roughness`	Optional coefficient-space roughness matrix. This bypasses field-space construction and is stored directly as Q.
`center`	Logical; if `TRUE`, center data before projection.
`ridge`	Small positive ridge added to the Gram diagonal if needed.
`label`	Optional label stored in metadata.
`...`	Additional arguments passed to the underlying lift path.

Value

An `AWPTBasisTemplate`.

`awpt_mean_conductance`

Average subject conductance matrices on a shared template graph

---

Description

Average subject conductance matrices on a shared template graph

Usage

```
awpt_mean_conductance(
  conductances,
  method = c("log_euclidean", "arithmetic"),
  shrinkage = 0,
  enforce_psd = TRUE,
  tol = 1e-08
)
```

Arguments

`conductances`	List of symmetric conductance matrices on the same template graph.
`method`	Averaging rule.
`shrinkage`	Optional shrinkage toward the isotropic identity.
`enforce_psd`	Logical; if `TRUE`, project the result to the PSD cone.
`tol`	Numerical tolerance for SPD operations.

Value

A symmetric averaged conductance matrix.

`awpt_operator_from_conductance`

Build an AWPT field operator from a conductance matrix

Description

Build an AWPT field operator from a conductance matrix

Usage

```
awpt_operator_from_conductance(
  conductance,
  normalize = c("none", "sym", "rw"),
  tol = 1e-08
)
```

Arguments

`conductance`	Symmetric conductance matrix on the template graph.
`normalize`	Laplacian normalization convention.
`tol`	Numerical tolerance.

Value

A field-space roughness operator.

`awpt_operator_from_subject_conductances`
Build an AWPT field operator from subject conductance summaries

---

Description

Build an AWPT field operator from subject conductance summaries

Usage

```
awpt_operator_from_subject_conductances(
  conductances,
  mean_method = c("log_euclidean", "arithmetic"),
  normalize = c("none", "sym", "rw"),
  shrinkage = 0,
  enforce_psd = TRUE,
  tol = 1e-08
)
```

Arguments

`conductances`	List of subject conductance matrices on a shared template graph.
`mean_method`	Averaging rule for the conductance mean.
`normalize`	Laplacian normalization convention.
`shrinkage`	Optional shrinkage toward an isotropic baseline before Laplacian construction.
`enforce_psd`	Logical; if `TRUE`, project the mean conductance to the PSD cone.
`tol`	Numerical tolerance.

Value

A list with `conductance_mean` and `operator`.

`awpt_surface_basis_template` *Build a surface AWPT basis template*

Description

Build a surface AWPT basis template

Usage

```
awpt_surface_basis_template(
  geometry,
  basis_spec = basis_awpt_wavelet(),
  support = NULL,
  loadings = NULL,
  centers = NULL,
  anatomical_operator = NULL,
  conductance = NULL,
  coefficient_roughness = NULL,
  measure = NULL,
  center = FALSE,
  ridge = 1e-08,
  label = "surface_awpt_wavelet"
)
```

Arguments

`geometry`	A `neurosurf::SurfaceGeometry` or `neurosurf::SurfaceSet`.
`basis_spec`	An AWPT basis specification created by `basis_awpt_wavelet()`.
`support`	Surface support as vertex indices or a logical vector over all vertices.
`loadings`	Optional explicit surface decoder loadings.
`centers`	Optional support-local center indices used for automatic wave-packet construction.
`anatomical_operator`	Optional field-space roughness operator on the supported surface domain.
`conductance`	Optional conductance matrix on the supported surface graph.
`coefficient_roughness`	Optional coefficient-space roughness matrix.
`measure`	Optional support-aligned weighting or mass information.
`center`	Logical; if `TRUE`, center data before projection.
`ridge`	Small positive ridge added to the Gram diagonal if needed.
`label`	Optional label stored in metadata.

Value

A `SurfaceAWPTBasisTemplate`.

`basis` *Get the basis matrix (temporal components)*

Description

Extract the basis matrix from a latent space representation. For `LatentNeuroVec` objects, this returns the temporal basis matrix with dimensions (nTime x k) where k is the number of components.

Usage

```
basis(x, ...)

## S4 method for signature 'LatentNeuroSurfaceVector'
basis(x)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
basis(x)

## S4 method for signature 'BlockLatentNeuroVector'
basis(x)

## S4 method for signature 'LatentNeuroVec'
basis(x)
```

Arguments

`x`	An object containing a basis matrix (e.g., `LatentNeuroVec`)
`...`	Additional arguments (currently unused)

Value

The basis matrix (typically time x components)

Examples

```
mask <- neuroim2::LogicalNeuroVol(
  array(TRUE, dim = c(2, 2, 1)),
  neuroim2::NeuroSpace(c(2, 2, 1))
)
lvec <- LatentNeuroVec(
  basis = matrix(1:6, nrow = 3),
  loadings = matrix(seq_len(8) / 10, nrow = 4),
  space = neuroim2::NeuroSpace(c(2, 2, 1, 3)),
  mask = mask,
  expect_dense = TRUE
)
b_matrix <- basis(lvec)
```

```
dim(b_matrix)
```

`basis_asset` *Extract the basis asset behind a latent object*

Description

Extract the basis asset behind a latent object

Usage

```
basis_asset(x, ...)

## S4 method for signature 'LatentNeuroSurfaceVector'
basis_asset(x, ...)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
basis_asset(x, ...)

## S4 method for signature 'ImplicitLatent'
basis_asset(x, ...)

## S4 method for signature 'LatentNeuroVec'
basis_asset(x, ...)

## S4 method for signature 'BlockLatentNeuroVector'
basis_asset(x, ...)
```

Arguments

`x`	A latent object.
`...`	Additional arguments passed to methods.

Value

Basis asset object or `NULL`.

basis_awpt_wavelet *AWPT wave-packet basis specification*

Description

AWPT wave-packet basis specification

Usage

```
basis_awpt_wavelet(
  scales = c(1, 2, 4, 8),
  order = 30L,
  threshold = 1e-06,
  k_neighbors = 6L,
  penalty_rule = c("inverse_scale", "inverse_scale_sq", "scale", "none", "custom"),
  custom_weights = NULL
)
```

Arguments

scales	Numeric vector of anatomical wave-packet scales.
order	Polynomial approximation order for the underlying heat-wavelet construction.
threshold	Threshold for small coefficients.
k_neighbors	Graph neighborhood parameter.
penalty_rule	Rule used to convert scales into roughness weights.
custom_weights	Optional explicit weights matching scales.

Value

A spec_awpt_wavelet object.

basis_decoder *Extract a basis decoder from a template asset*

Description

Extract a basis decoder from a template asset

Usage

```
basis_decoder(template, ...)

## S4 method for signature 'SurfaceAWPTBasisTemplate'
basis_decoder(template, ...)

## S4 method for signature 'AWPTBasisTemplate'
basis_decoder(template, ...)

## S4 method for signature 'HierarchicalBasisTemplate'
basis_decoder(template, ...)

## S4 method for signature 'ParcelBasisTemplate'
basis_decoder(template, ...)

## S4 method for signature 'SurfaceBasisTemplate'
basis_decoder(template, ...)
```

Arguments

`template`	A template object.
`...`	Additional arguments passed to methods.

Value

Basis decoder view object.

`basis_diffusion_wavelet` *Diffusion wavelet basis specification*

Description

Diffusion wavelet basis specification

Usage

```
basis_diffusion_wavelet(
  target_rank = 2000L,
  oversample = 20L,
  threshold = NULL,
  max_scales = 1L,
  epsilon = NULL,
  seed = 1L,
  sparsify_eps = NULL
)
```

Arguments

`target_rank`	Cap on retained components per scale (keeps runtime bounded).
`oversample`	Oversampling for randomized range finder.
`threshold`	Deprecated alias for 'sparsify_eps'.
`max_scales`	Maximum diffusion scales to compute.
`epsilon`	Optional precision (unused in capped-rank path; kept for API parity).
`seed`	Optional integer seed for deterministic randomized range finding.
`sparsify_eps`	Absolute value threshold to enforce sparsity in compressed operators. Stored as 'threshold' for compatibility.

Value

A 'spec_diffusion_wavelet' basis specification descriptor.

`basis_flat` *Create a flat basis specification*

Description

'basis_flat()' creates a descriptor for methods that should lift a reduction without estimating local components, typically as a simple parcel/cluster indicator basis.

Usage

```
basis_flat()
```

Value

An empty list with class 'spec_flat'.

`basis_heat_wavelet` *Heat wavelet basis specification*

Description

Heat wavelet basis specification

Usage

```
basis_heat_wavelet(scales = c(1, 2, 4, 8), order = 30, threshold = 1e-06)
```

Arguments

`scales`	Numeric vector of heat diffusion scales.
`order`	Polynomial approximation order.
`threshold`	Threshold for small coefficients.

Value

A ‘spec_heat_wavelet‘ object for ‘lift()‘.

`basis_pca` *Create a PCA basis specification*

Description

‘basis_pca()‘ creates a lightweight descriptor for parcel- or cluster-local PCA bases. The descriptor is consumed by ‘lift(ClusterReduction, spec_pca, data = ...)‘, where ‘data‘ supplies the time-by-voxel matrix used to estimate the components.

Usage

```
basis_pca(k = 3, whiten = FALSE)
```

Arguments

`k`	Positive integer number of PCA components.
`whiten`	Logical scalar. ‘basis_pca()‘ records this request, but ‘lift(ClusterReduction, spec_pca)‘ returns unwhitened loadings and emits a warning when ‘whiten = TRUE‘; the higher-level ‘encode(spec_space_pca())‘ path applies whitening after projection.

Value

A list with class ‘spec_pca‘ containing ‘k‘ and ‘whiten‘.

basis_slepian *Create a Slepian basis specification*

Description

'basis_slepian()' creates a lightweight descriptor for graph/Slepian basis construction during 'lift()' or spatial encoding. It records the requested component count and Slepian flavor; the actual basis is computed later by a 'lift()' method for the supplied reduction.

Usage

```
basis_slepian(k = 3, type = "laplacian")
```

Arguments

k	Positive integer number of Slepian components.
type	Character scalar naming the Slepian basis flavor. The built-in spatial methods use '"laplacian"'.

Value

A list with class 'spec_slepian' containing 'k' and 'type'.

benchmark_roundtrip *Benchmark encode/decode round-trips*

Description

Benchmark encode/decode round-trips

Usage

```
benchmark_roundtrip(
  mask_dims = c(16, 16, 8),
  n_time = 10L,
  methods = c("slepian_space", "hrbf", "wavelet_active", "bspline_hrbf_st"),
  iterations = 1L
)
```

Arguments

mask_dims	Integer vector length 3 for spatial dims.
n_time	Integer time points.
methods	Character vector of families: "slepian_space", "hrbf", "wavelet_active", "bspline_hrbf_st".
iterations	Integer iterations per method (default 1 to stay fast).

Value

A data.frame/tibble with timings and reconstruction error.

`BilatLatentNeuroSurfaceVector`
Construct a bilateral surface latent object

Description

Construct a bilateral surface latent object

Usage

```
BilatLatentNeuroSurfaceVector(left, right, label = "", meta = list())
```

Arguments

`left`	Left `LatentNeuroSurfaceVector`.
`right`	Right `LatentNeuroSurfaceVector`.
`label`	Optional label.
`meta`	Optional metadata list.

Value

A `BilatLatentNeuroSurfaceVector`.

`BilatLatentNeuroSurfaceVector-class`
BilatLatentNeuroSurfaceVector Class

Description

An explicit bilateral surface latent representation composed of left and right `LatentNeuroSurfaceVector` objects.

Slots

`left` Left-hemisphere latent surface object.

`right` Right-hemisphere latent surface object.

`label` Character label.

`meta` Lightweight metadata list.

`BlockLatentNeuroVector`
Construct a block-domain latent object

Description

Construct a block-domain latent object

Usage

```
BlockLatentNeuroVector(blocks, label = "", meta = list())
```

Arguments

`blocks`	Named list of explicit latent objects sharing a common basis.
`label`	Optional label.
`meta`	Optional metadata list.

Value

A `BlockLatentNeuroVector`.

`BlockLatentNeuroVector-class`
BlockLatentNeuroVector Class

Description

An explicit block-domain latent representation composed of multiple named latent blocks sharing a common coefficient basis. Intended for hybrid domains such as bilateral cortical surfaces plus volumetric subcortex.

Slots

`blocks` Named list of explicit latent objects.

`label` Character label.

`meta` Lightweight metadata list.

boldzip_events *Construct sparse innovation event settings for BOLDZip-SR*

Description

Construct sparse innovation event settings for BOLDZip-SR

Usage

```
boldzip_events(max_events = 256L, threshold_sd = 3, paired_fraction = 0.25)
```

Arguments

max_events	Maximum number of residual events to store.
threshold_sd	Residual event threshold in robust standard deviations.
paired_fraction	Minimum paired split amplitude agreement.

Value

A 'BoldZipSREvents' settings object.

boldzip_graph_spatial_basis *Build a spectral graph spatial basis for BOLDZip-SR*

Description

Constructs an experimental graph-adapted spatial basis from a symmetric adjacency matrix. Low-frequency graph Laplacian eigenvectors become coarse carrier support functions and the following higher-frequency eigenvectors become detail atoms. This is a materialized small-to-medium graph MVP, not a production graph-wavelet operator.

Usage

```
boldzip_graph_spatial_basis(
  adjacency,
  n_coarse = 8L,
  n_detail = NULL,
  normalized = TRUE,
  label = NULL
)
```

Arguments

`adjacency`	Symmetric non-negative adjacency matrix.
`n_coarse`	Number of low-frequency eigenvectors to use as 'phi_c'.
`n_detail`	Number of following eigenvectors to use as 'phi_d'. Defaults to all remaining eigenvectors.
`normalized`	Whether to use the normalized graph Laplacian.
`label`	Optional label stored in metadata.

Value

A 'BoldZipSRSpatialBasis' object.

`boldzip_parcel_reconstruct` *Reconstruct a matrix from parcel-average time series*

Description

Reconstruct a matrix from parcel-average time series

Usage

```
boldzip_parcel_reconstruct(X, parcels)
```

Arguments

`X`	Numeric matrix with dimensions 'voxels x time'.
`parcels`	Integer, factor, or character vector with one parcel label per row of 'X'.

Value

Matrix with the same dimensions as 'X', expanded from parcel means.

boldzip_quantization *Construct reliability-aware quantization settings for BOLDZip-SR*

Description

Construct reliability-aware quantization settings for BOLDZip-SR

Usage

```
boldzip_quantization(base_step = 0, epsilon = 1e-06)
```

Arguments

base_step	Base quantization step. Set to 0 to disable quantization.
epsilon	Small positive value used in reliability-shaped step sizes.

Value

A ‘BoldZipSRQuantization‘ settings object.

boldzip_reliability *Construct split-half reliability settings for BOLDZip-SR*

Description

Construct split-half reliability settings for BOLDZip-SR

Usage

```
boldzip_reliability(
  split = c("odd_even", "halves"),
  min_texture_reliability = 0,
  min_temporal_reliability = 0
)
```

Arguments

split	Split strategy. ‘"odd_even"‘ pairs adjacent odd/even time points; ‘"halves"‘ pairs the first and second half of the run.
min_texture_reliability	Minimum held-out reliability required to keep a fine-detail texture loading.
min_temporal_reliability	Minimum carrier reliability required before temporal coefficients are kept. Carriers below this threshold are zeroed.

Value

A 'BoldZipSRReliability' settings object.

`boldzip_spatial_basis` *Build a matrix spatial basis descriptor for BOLDZip-SR*

Description

Build a matrix spatial basis descriptor for BOLDZip-SR

Usage

```
boldzip_spatial_basis(
  phi_c = NULL,
  phi_d = NULL,
  label = NULL,
  basis_asset = NULL
)
```

Arguments

`phi_c`	Optional coarse basis matrix with rows equal to voxels.
`phi_d`	Optional detail basis matrix with rows equal to voxels. If 'NULL', the detail basis is the identity basis.
`label`	Optional label stored in metadata.
`basis_asset`	Optional source template or shared-basis object used to build this descriptor.

Value

A 'BoldZipSRSpatialBasis' object.

`boldzip_sr_decode` *Decode an experimental BOLDZip-SR object*

Description

Decode an experimental BOLDZip-SR object

Usage

```
boldzip_sr_decode(object, time_idx = NULL, roi = NULL)
```

Arguments

`object`	A ‘BoldZipSR‘ object.
`time_idx`	Optional integer time indices to return.
`roi`	Optional integer or logical row subset to return.

Value

Reconstructed matrix with rows as voxels/grayordinates and columns as time points.

`boldzip_sr_encode` *Encode a matrix with experimental BOLDZip-SR compression*

Description

‘boldzip_sr_encode()‘ is an experimental matrix-level implementation of Split-Reliable Graph Carrier Compression. It expects a matrix with rows as voxels or grayordinates and columns as time points. The implementation stores temporally compressed carriers, sparse high-resolution texture loadings, and sparse split-reliable residual events. It is intended as a research prototype, not a production binary codec.

Usage

```
boldzip_sr_encode(
  X,
  spatial_basis = NULL,
  k_carriers = 96L,
  temporal_k = NULL,
  temporal_spec = NULL,
  q_texture = 2L,
  texture_lags = 0L,
  reliability = boldzip_reliability(),
  quantization = boldzip_quantization(),
  events = boldzip_events(),
  center = TRUE,
  label = NULL
)
```

Arguments

`X`	Numeric matrix with dimensions ‘voxels x time‘.
`spatial_basis`	Optional ‘BoldZipSRSpatialBasis‘ object or list with ‘phi_c‘ and ‘phi_d‘ matrices. If omitted, the detail basis is identity and carriers are learned from ‘X‘ directly.
`k_carriers`	Number of carrier time courses to learn.
`temporal_k`	Number of DCT coefficients per carrier. Defaults to ‘ceiling(n_time / 4)‘ when ‘temporal_spec‘ is ‘NULL‘.

`temporal_spec`	Optional temporal basis descriptor. May be a 'SharedTemporalSpec', 'spec_time_dct', 'spec_time_bspline', or numeric basis matrix with rows equal to time points. When supplied, it determines the temporal basis and 'temporal_k'.
`q_texture`	Maximum number of carrier loadings per detail atom.
`texture_lags`	Integer vector of allowed carrier lags for texture loading fits. A positive lag uses 'Z_k(t - lag)'.
`reliability`	Reliability settings from [boldzip_reliability()].
`quantization`	Quantization settings from [boldzip_quantization()].
`events`	Event settings from [boldzip_events()].
`center`	Whether to store and remove a voxel-wise mean before fitting.
`label`	Optional label stored in metadata.

Value

A 'BoldZipSR' object.

`boldzip_sr_payload_summary`
Summarize an experimental BOLDZip-SR payload

Description

Summarize an experimental BOLDZip-SR payload

Usage

```
boldzip_sr_payload_summary(object)
```

Arguments

`object`	A 'BoldZipSR' object.

Value

Data frame with component counts and approximate object bytes.

`boldzip_sr_simulate` *Simulate data with BOLDZip-SR carrier, texture, and event structure*

Description

Simulate data with BOLDZip-SR carrier, texture, and event structure

Usage

```
boldzip_sr_simulate(
  n_voxels = 40L,
  n_time = 80L,
  k_carriers = 3L,
  q_texture = 1L,
  n_events = 8L,
  noise_sd = 0.05,
  seed = NULL
)
```

Arguments

`n_voxels`	Number of spatial rows.
`n_time`	Number of time points.
`k_carriers`	Number of latent carrier time courses.
`q_texture`	Number of non-zero carrier loadings per voxel.
`n_events`	Number of paired impulse events to add.
`noise_sd`	Independent Gaussian noise standard deviation.
`seed`	Optional random seed.

Value

List containing ‘X‘, ‘mu‘, ‘carriers‘, ‘texture‘, and ‘events‘.

`boldzip_svd_reconstruct` *Reconstruct a matrix from a low-rank SVD baseline*

Description

Reconstruct a matrix from a low-rank SVD baseline

Usage

```
boldzip_svd_reconstruct(X, rank, center = TRUE)
```

Arguments

`X`	Numeric matrix with dimensions ‘voxels x time‘.
`rank`	SVD rank.
`center`	Whether to remove and restore row means.

Value

Matrix with the same dimensions as ‘X‘.

`bspline_basis_handle` *Create a BasisHandle for a B-spline temporal basis*

Description

Create a BasisHandle for a B-spline temporal basis

Usage

```
bspline_basis_handle(
  n_time,
  k,
  degree = 3L,
  knots = NULL,
  boundary_knots = NULL,
  include_intercept = FALSE,
  orthonormalize = TRUE,
  id = NULL,
  label = NULL
)
```

Arguments

`n_time`	Integer, number of time points.
`k`	Integer, number of spline basis functions (df).
`degree`	Degree of the spline (default 3).
`knots`	Optional interior knots (scaled 0-1).
`boundary_knots`	Optional boundary knots (scaled 0-1).
`include_intercept`	Logical; include intercept column (default FALSE).
`orthonormalize`	Logical; orthonormalize columns when materialized.
`id`	Optional registry key (generated if NULL).
`label`	Optional human-readable label.

Value

A `BasisHandle`.

`build_hierarchical_template` *Build a hierarchical Laplacian template (offline)*

Description

Constructs a multi-level spatial basis from nested parcellations using graph Laplacian eigenvectors. The resulting template can be reused to efficiently encode multiple datasets that share the same mask geometry.

Usage

```
build_hierarchical_template(
  mask,
  parcellations,
  k_per_level,
  k_neighbors = 6L,
  ridge = 1e-08,
  solver = c("chol", "qr"),
  label = "hierarchical_laplacian"
)
```

Arguments

`mask`	LogicalNeuroVol or logical array (3D) defining the domain.
`parcellations`	List of integer vectors (one per level) of length = #voxels in mask. Levels must be nested: each child parcel maps to exactly one parent parcel in the previous level.
`k_per_level`	Integer vector giving #modes per parcel at each level.
`k_neighbors`	k for local graph construction inside parcels.
`ridge`	Small diagonal ridge added to $G = t(B)\% * \%B$ for stability.
`solver`	Solver choice: "chol" (default) or "qr" fallback.
`label`	Optional label stored in meta.

Value

HierarchicalBasisTemplate (primal basis B + cached solver and metadata).

`build_schaefer_hierarchical_template` *Build hierarchical template from Schaefer surface atlas*

Description

Convenience wrapper that combines 'build_schaefer_levels()' with 'build_hierarchical_template()' to produce a ready-to-use template with geodesic-informed parcel clustering.

Usage

```
build_schaefer_hierarchical_template(
  mask,
  parcels = 400,
  networks = 17,
  space = "fsaverage6",
  k_levels = c(16, 32, 64),
  k_per_level = c(8, 5, 3, 1),
  vol_atlas = NULL,
  alpha = 0.5,
  beta = 0.3,
  gamma = 0.2,
  d0 = 30,
  component_policy = "largest",
  cache = TRUE,
  k_neighbors = 6L,
  ridge = 1e-08,
  solver = c("chol", "qr")
)
```

Arguments

`mask`	LogicalNeuroVol defining the 3D brain mask (MNI space).
`parcels`	Integer. Number of Schaefer parcels (100, 200, 300, 400, 500, 600, 800, 1000).
`networks`	Integer. Yeo network variant (7 or 17).
`space`	Character. Surface space ("fsaverage6", "fsaverage", "fsaverage5").
`k_levels`	Integer vector of cluster counts for coarser levels, ordered coarse→fine. The finest level is always the original Schaefer parcellation. Example: c(16, 32, 64) produces 4 levels: L0=16, L1=32, L2=64, L3=Schaefer-parcels.
`k_per_level`	Integer vector of eigenmodes per parcel at each level. Length must match length(k_levels) + 1 (for the finest Schaefer level).
`vol_atlas`	Optional. Pre-loaded volumetric Schaefer atlas (NeuroVol with parcel IDs). If NULL, attempts to load via neuroatlas::get_schaefer_atlas().

alpha	Weight for boundary contact in similarity (default 0.5).
beta	Weight for geodesic kernel in similarity (default 0.3).
gamma	Weight for Yeo network agreement in similarity (default 0.2).
d0	Scale for geodesic exponential kernel (default 30 mm).
component_policy	How to handle fragmented parcels ("error", "largest", "each", "merge").
cache	Logical. Use cached geodesic distances if available (default TRUE).
k_neighbors	k for local graph construction inside parcels.
ridge	Small diagonal ridge for Gram matrix stability.
solver	Solver choice: "chol" or "qr".

Value

HierarchicalBasisTemplate object.

See Also

build_hierarchical_template, build_schaefer_levels

---

build_schaefer_levels

Build hierarchical parcellation levels from Schaefer surface atlas

---

Description

Constructs nested parcellation levels for the hierarchical template by: 1. Loading Schaefer surface atlas from neuroatlas 2. Computing geodesic distances and boundary contacts via neurosurf 3. Building a parcel similarity matrix (geodesic + boundary + network) 4. Performing spectral Ward clustering to create coarser levels 5. Mapping surface parcels back to volumetric voxel labels

Real-data glue to finish: - Provide fsaverage Schaefer surfaces (via 'neuroatlas::schaefer_surf') so we can compute geodesic/boundary terms. - Provide the volumetric Schaefer atlas aligned to the MNI mask (via 'neuroatlas::get_schaefer_atlas'). - Ensure mask and volumetric atlas share resolution/origin/spacing; this function assumes 2 mm MNI unless you pass a custom 'vol_atlas'. - Subcortex is **not** handled here; add it separately when assembling the full template.

Usage

```
build_schaefer_levels(
  mask,
  parcels = 400,
  networks = 17,
  space = "fsaverage6",
  k_levels = c(16, 32, 64),
```

```
  vol_atlas = NULL,
  alpha = 0.5,
  beta = 0.3,
  gamma = 0.2,
  d0 = 30,
  component_policy = c("largest", "error", "each", "merge"),
  cache = TRUE
)
```

Arguments

mask	LogicalNeuroVol defining the 3D brain mask (MNI space).
parcels	Integer. Number of Schaefer parcels (100, 200, 300, 400, 500, 600, 800, 1000).
networks	Integer. Yeo network variant (7 or 17).
space	Character. Surface space ("fsaverage6", "fsaverage", "fsaverage5").
k_levels	Integer vector of cluster counts for coarser levels, ordered coarse→fine. The finest level is always the original Schaefer parcellation. Example: c(16, 32, 64) produces 4 levels: L0=16, L1=32, L2=64, L3=Schaefer-parcels.
vol_atlas	Optional. Pre-loaded volumetric Schaefer atlas (NeuroVol with parcel IDs). If NULL, attempts to load via neuroatlas::get_schaefer_atlas().
alpha	Weight for boundary contact in similarity (default 0.5).
beta	Weight for geodesic kernel in similarity (default 0.3).
gamma	Weight for Yeo network agreement in similarity (default 0.2).
d0	Scale for geodesic exponential kernel (default 30 mm).
component_policy	How to handle fragmented parcels ("error", "largest", "each", "merge").
cache	Logical. Use cached geodesic distances if available (default TRUE).

Details

Requires 'neuroatlas' and 'neurosurf' packages. The surface atlas provides geodesic distances and boundary topology; the volumetric atlas maps parcels to mask voxels.

Value

A list with components:

levels	List of integer vectors (length = #voxels in mask), one per level, nested.
network	Character vector of Yeo network labels per finest-level parcel.
hemi	Character vector of hemisphere ("L"/"R") per finest-level parcel.
geo_dist_lh, geo_dist_rh	Parcel geodesic distance matrices (for diagnostics).
boundary_lh, boundary_rh	Boundary contact matrices (LH/RH).

`ClusterReduction-class` *Cluster-based reduction (e.g., supervoxels or atlas)*

Description

Cluster-based reduction (e.g., supervoxels or atlas)

Slots

`map` Integer vector (voxel order) mapping each voxel to a cluster id.

`cluster_ids` Unique cluster ids present in ‘map‘.

`CoarsenedReduction-class` *Coarsened graph reduction (e.g., prolongation from coarse to fine)*

Description

Coarsened graph reduction (e.g., prolongation from coarse to fine)

Slots

`P_matrix` Sparse prolongation matrix (fine x coarse).

`coarse_adj` Optional sparse adjacency on the coarse graph.

`coef_metric` *Extract coefficient-space metric from a latent object*

Description

Extract coefficient-space metric from a latent object

Usage

```
coef_metric(x, coordinates = c("analysis", "raw"), ...)

## S4 method for signature 'LatentNeuroSurfaceVector'
coef_metric(x, coordinates = c("analysis", "raw"), ...)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
coef_metric(x, coordinates = c("analysis", "raw"), ...)

## S4 method for signature 'ImplicitLatent'
coef_metric(x, coordinates = c("analysis", "raw"), ...)

## S4 method for signature 'LatentNeuroVec'
coef_metric(x, coordinates = c("analysis", "raw"), ...)

## S4 method for signature 'BlockLatentNeuroVector'
coef_metric(x, coordinates = c("analysis", "raw"), ...)
```

Arguments

`x`	A latent object.
`coordinates`	Coordinate system for the requested metric.
`...`	Additional arguments passed to methods.

Value

Matrix-like metric representation or `NULL`. For transport-backed latent objects, analysis coordinates are Euclidean by contract in v1. Raw-coordinate metrics are returned when the raw-to-analysis transform exposes a linear matrix representation.

`coef_time` *Extract coefficient time series from a latent object*

Description

Extract coefficient time series from a latent object

Usage

```
coef_time(x, coordinates = c("analysis", "raw"), ...)

## S4 method for signature 'LatentNeuroSurfaceVector'
coef_time(x, coordinates = c("analysis", "raw"), ...)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
coef_time(x, coordinates = c("analysis", "raw"), ...)
```

```
## S4 method for signature 'ImplicitLatent'
coef_time(x, coordinates = c("analysis", "raw"), ...)

## S4 method for signature 'LatentNeuroVec'
coef_time(x, coordinates = c("analysis", "raw"), ...)

## S4 method for signature 'BlockLatentNeuroVector'
coef_time(x, coordinates = c("analysis", "raw"), ...)
```

Arguments

`x`	A latent object.
`coordinates`	Coordinate system for the returned coefficients.
`...`	Additional arguments passed to methods.

Value

Numeric matrix with rows = time and columns = latent coefficients.

`compare_boldzip_sr` *Compare BOLDZip-SR against simple reconstruction baselines*

Description

Compare BOLDZip-SR against simple reconstruction baselines

Usage

```
compare_boldzip_sr(X, fit, parcels = NULL, svd_ranks = NULL)
```

Arguments

`X`	Original matrix with rows as voxels/grayordinates and columns as time points.
`fit`	A ‘BoldZipSR‘ object.
`parcels`	Optional parcel labels for a parcel-average baseline.
`svd_ranks`	Optional integer vector of SVD ranks.

Value

Data frame with method, MSE, RMSE, correlation, and scalar payload estimates.

`concat,LatentNeuroVec,LatentNeuroVec-method`
Concatenate LatentNeuroVec Objects

Description

Concatenates two or more `LatentNeuroVec` objects along the temporal dimension.

Usage

```
## S4 method for signature 'LatentNeuroVec,LatentNeuroVec'
concat(x, y, ...)
```

Arguments

`x`	First `LatentNeuroVec`.
`y`	Second `LatentNeuroVec`.
`...`	Additional `LatentNeuroVec` objects to concatenate.

Value

A new `LatentNeuroVec` if all objects are compatible, otherwise a `NeuroVecSeq-class`.

Examples

```
mask <- neuroim2::LogicalNeuroVol(
  array(TRUE, dim = c(2, 2, 1)),
  neuroim2::NeuroSpace(c(2, 2, 1))
)
lvec <- LatentNeuroVec(
  basis = matrix(1:6, nrow = 3),
  loadings = matrix(seq_len(8) / 10, nrow = 4),
  space = neuroim2::NeuroSpace(c(2, 2, 1, 3)),
  mask = mask,
  expect_dense = TRUE
)
combined <- concat(lvec, lvec)
dim(combined)
```

cut_hclust_nested *Cut an hclust into nested label vectors*

Description

Utility to turn a hierarchical clustering (on parcels) into a nested set of parcel label vectors that downstream code can lift into voxel space. Real data glue: you will pass the hclust built on a parcel-level similarity graph (e.g., Schaefer-400 parcel graph constructed from surface geodesics and boundary contacts); then map these parcel labels back to voxels using the volumetric atlas.

Usage

```
cut_hclust_nested(hc, k_levels)
```

Arguments

hc	hclust object (e.g., from spectral_ward_hclust()).
k_levels	Integer vector of cluster counts, ordered coarse→fine (e.g., c(16, 32, 64, 400)).

Value

List of integer label vectors (same length as hc$labels), one per level, nested by construction.

dct_basis_handle *Create a BasisHandle for a DCT temporal basis*

Description

Create a BasisHandle for a DCT temporal basis

Usage

```
dct_basis_handle(n_time, k, norm = c("ortho", "none"), id = NULL, label = NULL)
```

Arguments

n_time	Number of time points.
k	Number of components.
norm	Normalization passed to [build_dct_basis()].
id	Optional registry key; if NULL a deterministic id is derived from parameters.
label	Optional human-readable label.

Value

A `BasisHandle` object.

`decode_coefficients` *Decode coefficient-space vectors into an output space*

Description

Decode coefficient-space vectors into an output space

Usage

```
decode_coefficients(
  x,
  gamma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  wrap = c("none", "auto"),
  ...
)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
decode_coefficients(
  x,
  gamma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  wrap = c("none", "auto"),
  ...
)

## S4 method for signature 'ImplicitLatent'
decode_coefficients(
  x,
  gamma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  wrap = c("none", "auto"),
  ...
)

## S4 method for signature 'LatentNeuroSurfaceVector'
decode_coefficients(
  x,
  gamma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
```

```
  wrap = c("none", "auto"),
  ...
)

## S4 method for signature 'LatentNeuroVec'
decode_coefficients(
  x,
  gamma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  wrap = c("none", "auto"),
  ...
)

## S4 method for signature 'BlockLatentNeuroVector'
decode_coefficients(
  x,
  gamma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  wrap = c("none", "auto"),
  ...
)
```

Arguments

`x`	A latent object.
`gamma`	Coefficient-space vector or matrix.
`space`	Output space to decode into.
`coordinates`	Coordinate system used by `gamma`.
`wrap`	If `"auto"`, wrap the decoded values with `wrap_decoded()` so the result is domain-aware (for example a `NeuroVol` for volumetric targets or a surface field for surface targets). Defaults to `"none"`, which returns the raw numeric vector or matrix.
`...`	Additional arguments passed to methods.

Value

Numeric vector or matrix in the requested output space, or a domain-aware wrapper when `wrap = "auto"`.

`decode_covariance` *Push coefficient covariance into an output space*

Description

Push coefficient covariance into an output space

Usage

```
decode_covariance(
  x,
  Sigma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  diag_only = TRUE,
  ...
)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
decode_covariance(
  x,
  Sigma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  diag_only = TRUE,
  ...
)

## S4 method for signature 'ImplicitLatent'
decode_covariance(
  x,
  Sigma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  diag_only = TRUE,
  ...
)

## S4 method for signature 'LatentNeuroSurfaceVector'
decode_covariance(
  x,
  Sigma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  diag_only = TRUE,
  ...
)

## S4 method for signature 'LatentNeuroVec'
decode_covariance(
  x,
  Sigma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  diag_only = TRUE,
  ...
```

```
)

## S4 method for signature 'BlockLatentNeuroVector'
decode_covariance(
  x,
  Sigma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  diag_only = TRUE,
  ...
)
```

Arguments

`x`	A latent object.
`Sigma`	Coefficient covariance matrix.
`space`	Output space to decode into.
`coordinates`	Coordinate system used by `Sigma`.
`diag_only`	Logical; if `TRUE`, return only the diagonal.
`...`	Additional arguments passed to methods.

Value

Numeric vector or matrix in the requested output space.

`decoder` *Get a decoder view for a latent object*

Description

Get a decoder view for a latent object

Usage

```
decoder(
  x,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  ...
)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
decoder(
  x,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
```

```
  ...
)

## S4 method for signature 'ImplicitLatent'
decoder(
  x,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  ...
)

## S4 method for signature 'LatentNeuroSurfaceVector'
decoder(
  x,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  ...
)

## S4 method for signature 'LatentNeuroVec'
decoder(
  x,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  ...
)

## S4 method for signature 'BlockLatentNeuroVector'
decoder(
  x,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  ...
)
```

Arguments

`x`	A latent object.
`space`	Output space to decode into.
`coordinates`	Coordinate system consumed by the decoder.
`...`	Additional arguments passed to methods.

Value

Decoder view object.

`diffusion_wavelet_latent` *Diffusion wavelet latent constructor (explicit basis)*

Description

Diffusion wavelet latent constructor (explicit basis)

Usage

```
diffusion_wavelet_latent(
  X,
  mask,
  reduction = NULL,
  spec = basis_diffusion_wavelet(),
  k_neighbors = 6L,
  label = ""
)
```

Arguments

`X`	Matrix time x voxels (mask order).
`mask`	LogicalNeuroVol or 3D logical array.
`reduction`	ClusterReduction; if NULL, defaults to one cluster per voxel.
`spec`	diffusion wavelet spec (basis_diffusion_wavelet()).
`k_neighbors`	k for graph building.
`label`	Optional label.

Value

A 'LatentNeuroVec' object.

`diffusion_wavelet_loadings_handle` *Construct a shared LoadingsHandle via diffusion-wavelet lifting*

Description

Wraps a diffusion-wavelet 'lift()' call so multiple 'LatentNeuroVec' instances can share the same spatial dictionary without embedding the full matrix in each object.

Usage

```
diffusion_wavelet_loadings_handle(
  reduction,
  basis_spec = basis_diffusion_wavelet(),
  data = NULL,
  k_neighbors = 6L,
  id = NULL,
  label = "diffusion-wavelet"
)
```

Arguments

`reduction`	Graph/cluster reduction used by ‘lift()‘.
`basis_spec`	Basis specification; defaults to ‘basis_diffusion_wavelet()‘.
`data`	Ignored by diffusion-wavelet lifting; accepted only to keep the lifted-handle constructor signature aligned across families.
`k_neighbors`	k for graph building when lifting.
`id`	Optional registry id; provide a stable string to reuse across sessions. If NULL, a deterministic id is derived from the spec and reduction.
`label`	Optional human-readable label.

Value

A `LoadingsHandle`.

`dpss_time_basis` *DPSS temporal basis (Slepian sequences)*

Description

Generates Discrete Prolate Spheroidal Sequences (DPSS) for a given series length and bandwidth. Uses an internal RcppEigen solver over the prolate matrix; suitable for moderate ‘n_time‘. For very long series, a tridiagonal solver can replace the backend with the same interface.

Usage

```
dpss_time_basis(
  n_time,
  tr,
  bandwidth,
  k = NULL,
  backend = c("tridiag", "dense")
)
```

Arguments

`n_time`	Integer length of the time dimension.
`tr`	Repetition time in seconds.
`bandwidth`	Half-bandwidth in Hz (typical BOLD range 0.008–0.1).
`k`	Optional number of tapers; defaults to `floor(2 * NW) - 1` where `NW = n_time * bandwidth * tr`. Clamped to `[1, n_time]`.
`backend`	DPSS backend. Only `"tridiag"` is currently supported.

Value

Numeric matrix (n_time x k) with orthonormal columns.

`encode` *Encode data into a latent representation using a spec*

Description

Encode data into a latent representation using a spec

Usage

```
encode(
  x,
  spec,
  mask,
  reduction = NULL,
  materialize = c("auto", "handle", "matrix"),
  label = "",
  ...
)
```

Arguments

`x`	Matrix (time x voxels in mask order).
`spec`	Standard encode spec object created by ‘spec_time_*‘, ‘spec_space_*‘, or ‘spec_st‘. AWPT specs created by [basis_awpt_wavelet()] describe shared templates and are intentionally not accepted by ‘encode()‘; use [encode_awpt()] or [encode_operator()] for transport-backed AWPT fits.
`mask`	LogicalNeuroVol or logical array (required for spatial pieces).
`reduction`	Optional GraphReduction (for spatial specs).
`materialize`	”handle”, ”matrix”, or ”auto” (default ”handle”).
`label`	Optional label.
`...`	Additional arguments passed to methods.

Value

The return class depends on the spec family:

Explicit spatial families ‘spec_space_slepian‘, ‘spec_space_heat‘, ‘spec_space_hrbf‘, ‘spec_space_pca‘, and ‘spec_space_wavelet_active‘ return a [LatentNeuroVec], which is a concrete ‘ExplicitLatent‘ (the virtual marker defined at ‘R/all_class.R‘). It stores an explicit ‘basis x loadings + offset‘ factorization.

Explicit temporal families ‘spec_time_slepian‘, ‘spec_time_dct‘, and ‘spec_time_bspline‘ likewise return a [LatentNeuroVec] (‘ExplicitLatent‘).

Spatiotemporal (‘spec_st‘) **Always** returns an ‘ImplicitLatent‘ (a decoder-only / separable representation), even when both the time and space factors are fully explicit dense bases. ‘ImplicitLatent‘ is *not* an ‘ExplicitLatent‘.

Transport (‘spec_transport‘ / AWPT encoders) return a ‘TransportLatent‘, which is also *not* an ‘ExplicitLatent‘.

In short: ‘ExplicitLatent‘ is the virtual S4 marker inherited by [LatentNeuroVec]; ‘ImplicitLatent‘ and ‘TransportLatent‘ are S3 classes that deliberately do not inherit it.

Dispatch model

For standard in-mask matrix encoders, ‘encode()‘ routes to the S3 generic [encode_spec()], which dispatches on the spec class and builds the latent object directly. Transport-backed AWPT is the explicit exception: an AWPT basis spec is used to build a shared template, while the subject fit also requires a ‘basis_asset‘ and ‘field_operator‘. Those assets are not part of the standard ‘encode_spec()‘ signature, so AWPT enters through [encode_awpt()] or [encode_operator()] instead of [encode()].

Offset and centering contract

A [LatentNeuroVec] reconstructs its data as ‘basis (length = number of in-mask voxels) added back after the low-rank term. The offset is owned by the encoder's *lift* step, which is the single place that decides whether and how to center:

Families that populate ‘offset‘ Only ‘spec_space_pca‘ produces a non-empty offset, and only when ‘center = TRUE‘ (the default): the per-voxel column means are removed before the PCA and stored in the ‘offset‘ slot so reconstruction restores them. The mean removal is performed exactly once, inside ‘lift.basis_pca‘ (see the ‘fmrilatent.mean_scores‘ attribute it returns); the encode caller does not re-center. With ‘center = FALSE‘, PCA stores ‘offset = numeric(0)‘.

Families that never center All other explicit families (‘spec_space_slepian‘, ‘spec_space_heat‘, ‘spec_space_hrbf‘, ‘spec_time_slepian‘, ‘spec_time_dct‘, ‘spec_time_bspline‘) store ‘offset = numeric(0)‘, i.e. no offset.

By convention ‘offset = numeric(0)‘ means "no offset" and reconstruction treats it as a zero vector; a populated ‘offset‘ always has one entry per in-mask voxel. The shared ‘.encode_center()‘ helper is the single implementation of column-mean centering used by the offset-producing paths.

`encode_awpt` *Encode data using the AWPT model*

Description

Encode data using the AWPT model

Usage

```
encode_awpt(
  x,
  basis_asset,
  field_operator = NULL,
  observation_operator = NULL,
  mask = NULL,
  domain = NULL,
  support = NULL,
  spatial_lambda = 0,
  temporal_lambda = 0,
  temporal_order = 1L,
  sparse_lambda = 0,
  sparse_mode = c("none", "group_l2", "lasso"),
  max_iter = 200L,
  tol = 1e-06,
  center = TRUE,
  run_info = NULL,
  label = "",
  ...
)
```

Arguments

`x`	Numeric matrix (time x target samples) or a `NeuroVec`.
`basis_asset`	An `AWPTBasisTemplate`.
`field_operator`	Subject field operator. See `encode_operator()` for the required contract.
`observation_operator`	Legacy alias for `field_operator`.
`mask`	Optional volumetric target mask for the field-operator target domain.
`domain`	Optional non-volumetric target domain.
`support`	Optional target support aligned to `domain`.
`spatial_lambda`	Strength of the anatomical roughness penalty.
`temporal_lambda`	Strength of temporal smoothing.

temporal_order	Temporal difference order for smoothing.
sparse_lambda	Strength of optional sparse atom selection.
sparse_mode	Sparse penalty mode. Use "group_l2" for atom-wise group shrinkage.
max_iter	Maximum iterations for sparse AWPT optimization.
tol	Relative convergence tolerance for sparse AWPT optimization.
center	Logical; if TRUE, center target samples before solving.
run_info	Optional run metadata; run_lengths control temporal blocks.
label	Optional label stored in metadata.
...	Reserved for future extensions.

Details

AWPT does not expose a separate ridge penalty. The returned metadata records lambda = 0; the anatomical roughness penalty is recorded separately as spatial_lambda.

Value

A TransportLatent object with AWPT metadata.

---

encode_hierarchical *Encode data using a hierarchical template*

Description

Note: unlike parcel and AWPT encoding, hierarchical encoding does not center the data. The returned offset is always numeric(0).

Usage

```
encode_hierarchical(
  X,
  template,
  label = NULL,
  mask = NULL,
  materialize = c("handle", "auto", "matrix")
)
```

Arguments

X	matrix time x voxels (mask order) matching template mask
template	HierarchicalBasisTemplate
label	Optional label for the resulting LatentNeuroVec (defaults to template label)
mask	Optional mask to validate against the template mask before encoding. When supplied, it must match the template exactly.
materialize	"handle", "matrix", or "auto" for template loadings.

Value

LatentNeuroVec with basis = time x atoms coefficients, loadings = template loadings

---

`encode_operator` *Encode data using a shared basis asset and subject field operator*

Description

Encode data using a shared basis asset and subject field operator

Usage

```
encode_operator(
  x,
  template,
  field_operator = NULL,
  observation_operator = NULL,
  mask = NULL,
  domain = NULL,
  support = NULL,
  lambda = 0,
  center = TRUE,
  run_info = NULL,
  spatial_lambda = lambda,
  spatial_penalty = NULL,
  temporal_lambda = 0,
  temporal_order = 1L,
  sparse_lambda = 0,
  sparse_mode = c("none", "group_l2", "lasso"),
  max_iter = 200L,
  tol = 1e-06,
  label = "",
  ...
)
```

Arguments

`x`	Numeric matrix (time x target samples) or a `NeuroVec`.
`template`	Shared basis asset providing `basis_decoder()`.
`field_operator`	Subject field operator mapping template field samples to observed native samples.
`observation_operator`	Legacy alias for `field_operator`.
`mask`	Optional volumetric target mask for the field-operator target domain.
`domain`	Optional non-volumetric target domain, for example a surface geometry.

`support` Optional target support aligned to `domain`, for example vertex indices on a surface. At least one of `mask` or `support` must be available either explicitly or via `field_operator$provenance`.

`lambda` Ridge penalty strength.

`center` Logical; if `TRUE`, center target samples before solving.

`run_info` Optional run metadata carried on the resulting latent object.

`spatial_lambda` Strength of the spatial coefficient penalty.

`spatial_penalty` Optional coefficient-space roughness matrix or diagonal weights.

`temporal_lambda` Strength of temporal smoothing.

`temporal_order` Difference order used for temporal smoothing.

`sparse_lambda` Strength of optional sparse coefficient shrinkage.

`sparse_mode` Sparse penalty mode. Use `"group_l2"` for atom-wise group shrinkage.

`max_iter` Maximum iterations for sparse AWPT optimization.

`tol` Relative convergence tolerance for sparse AWPT optimization.

`label` Optional label stored in metadata.

`...` Reserved for future extensions.

Details

The external field-operator contract is intentionally narrow. 'fmrilatent' consumes an operator-like object with:

`n_source`, `n_target` Source and target dimensions.

`source_domain_id`, `target_domain_id` Stable domain identifiers.

`forward(z)` Applies the operator to template field samples.

`adjoint_apply(y)` Applies the adjoint map.

`provenance$target_mask` Optional target-domain mask when the caller does not pass `mask` explicitly.

On the main quadratic and sparse transport/AWPT paths, coefficient recovery is matrix-free: 'fmrilatent' uses the operator's forward and adjoint methods rather than materializing the full subject decoder.

Value

A `TransportLatent` object.

encode_spec *Dispatch standard encoding based on spec type*

Description

Dispatch standard encoding based on spec type

Usage

```
encode_spec(x, spec, ...)
```

Arguments

x	Data matrix.
spec	Spec object.
...	Additional arguments passed to methods.

Details

‘encode_spec()‘ is the S3 dispatch seam for standard ‘encode()‘ specs: temporal, spatial, hierarchical, parcel, and spatiotemporal specs. AWPT’s ‘basis_awpt_wavelet()‘ object is a template-construction spec, not a complete subject-encoding spec, because AWPT fitting additionally needs a shared template/basis asset and subject field or observation operators. AWPT therefore uses the parallel transport API [encode_awpt()] / [encode_operator()].

Value

Encoded representation.

encode_transport *Encode data using transport-backed latent semantics*

Description

Encode data using transport-backed latent semantics

Usage

```
encode_transport(
  x,
  basis_asset,
  field_operator = NULL,
  observation_operator = NULL,
  mask = NULL,
  domain = NULL,
  support = NULL,
```

```
  lambda = 0,
  center = TRUE,
  run_info = NULL,
  spatial_lambda = lambda,
  spatial_penalty = NULL,
  temporal_lambda = 0,
  temporal_order = 1L,
  sparse_lambda = 0,
  sparse_mode = c("none", "group_l2", "lasso"),
  max_iter = 200L,
  tol = 1e-06,
  label = "",
  ...
)
```

Arguments

`x`	Numeric matrix (time x target samples) or a `NeuroVec`.
`basis_asset`	Shared basis asset.
`field_operator`	Subject field operator. See `encode_operator()` for the required contract.
`observation_operator`	Legacy alias for `field_operator`.
`mask`	Optional volumetric target mask for the field-operator target domain.
`domain`	Optional non-volumetric target domain.
`support`	Optional target support aligned to `domain`.
`lambda`	Ridge penalty strength.
`center`	Logical; if `TRUE`, center target samples before solving.
`run_info`	Optional run metadata carried on the resulting latent object.
`spatial_lambda`	Strength of the spatial coefficient penalty.
`spatial_penalty`	Optional coefficient-space roughness matrix or diagonal weights.
`temporal_lambda`	Strength of temporal smoothing.
`temporal_order`	Difference order used for temporal smoothing.
`sparse_lambda`	Strength of optional sparse coefficient shrinkage.
`sparse_mode`	Sparse penalty mode. Use `"group_l2"` for atom-wise group shrinkage.
`max_iter`	Maximum iterations for sparse AWPT optimization.
`tol`	Relative convergence tolerance for sparse AWPT optimization.
`label`	Optional label stored in metadata.
`...`	Reserved for future extensions.

Value

A `TransportLatent` object.

`evaluate_boldzip_sr` *Evaluate BOLDZip-SR reconstruction quality*

Description

Evaluate BOLDZip-SR reconstruction quality

Usage

```
evaluate_boldzip_sr(X, object, reliability_weights = NULL)
```

Arguments

`X`	Original matrix with rows as voxels/grayordinates and columns as time points.
`object`	A 'BoldZipSR' object or a reconstructed matrix.
`reliability_weights`	Optional matrix or vector of reliability weights.

Value

Named numeric vector with reconstruction metrics.

`fmrilatent_compat_profile`
Compatibility Profile for External Integrations

Description

Returns the active fmrilatent compatibility profile. This is used to opt into strict behavior for external packages that require historical semantics.

Usage

```
fmrilatent_compat_profile(profile = NULL)
```

Arguments

`profile`	Optional explicit profile name. If 'NULL', uses 'getOption("fmrilatent.compat", "native")'.

Value

A single character string profile identifier.

fmrilatent_registry_clear

Clear the fmrilatent handle registry

Description

Removes cached materialized matrices from the internal registry. Use this to free memory or force re-materialization of handles.

Usage

```
fmrilatent_registry_clear(type = c("all", "basis", "loadings"))
```

Arguments

type	Which registry to clear: "basis", "loadings", or "all" (default).

Value

Invisibly, the number of entries removed.

Examples

```
# Clear all cached matrices
fmrilatent_registry_clear()

# Clear only basis matrices
fmrilatent_registry_clear("basis")
```

fmrilatent_registry_enable

Enable or disable the fmrilatent handle registry

Description

The handle registry caches materialized matrices for `BasisHandle` and `LoadingsHandle` objects. This can improve performance and reduce memory duplication when multiple `LatentNeuroVec` objects share the same handle IDs.

Usage

```
fmrilatent_registry_enable()

fmrilatent_registry_disable()

fmrilatent_registry_enabled()
```

Details

Set `fmrilatent_registry_disable()` to turn off caching (useful for deterministic benchmarking or to avoid retaining large matrices).

Value

Invisibly, `TRUE`.

Capacity

The cache is bounded. Each registry (basis and loadings) holds at most `getOption("fmrilatent.registry.max_e 256)` materialized matrices; once full, the least-recently-used entry is evicted on the next registration. Both registration and retrieval count as a use. Set the option to `Inf` (or a non-positive value) to restore an unbounded cache, or call `fmrilatent_registry_clear()` to drop everything immediately.

Examples

```
fmrilatent_registry_disable()
fmrilatent_registry_enable()
```

---

fmrilatent_registry_list

List entries in the fmrilatent handle registry

Description

Returns the IDs of all cached matrices in the registry.

Usage

```
fmrilatent_registry_list(type = c("all", "basis", "loadings"))
```

Arguments

`type`	Which registry to list: "basis", "loadings", or "all" (default).

Value

Character vector of registered IDs.

Examples

```
# See what's cached
fmrilatent_registry_list()
```

fmrilatent_registry_stats *Get registry statistics*

Description

Returns count and approximate memory usage of cached matrices.

Usage

```
fmrilatent_registry_stats(type = c("all", "basis", "loadings"))
```

Arguments

type	Which registry: "basis", "loadings", or "all" (default).

Value

A list with count and bytes for each registry type.

Examples

```
fmrilatent_registry_stats()
```

fmrilatent_test_data *Generate test data for encoder development*

Description

Creates a small synthetic dataset suitable for testing `encode_spec` methods. Useful for extension packages that implement custom encoders.

Usage

```
fmrilatent_test_data(dims = c(3L, 3L, 2L), n_time = 8L)
```

Arguments

dims	Integer vector of spatial dimensions (default `c(3, 3, 2)`).
n_time	Number of time points (default 8).

Value

A list with elements:

X	Numeric matrix (n_time x n_voxels) of random data.
mask	Logical array of dimensions `dims`, all `TRUE`.
dims	The spatial dimensions used.
n_time	The number of time points used.

Examples

```
td <- fmrilatent_test_data()
dim(td$X)        # 8 x 18
dim(td$mask)     # 3 x 3 x 2
```

---

get_encoder *Get a registered encoder*

Description

Retrieves the registration entry for a given encoder family.

Usage

```
get_encoder(family)
```

Arguments

`family`	Character string identifying the encoder family.

Value

A list with elements `spec_fn`, `description`, and `package`.

Examples

```
enc <- get_encoder("time_dct")
spec <- enc$spec_fn(k = 5)
```

GraphReduction-class *Graph reduction scaffolds (abstract)*

Description

These classes describe how voxels are grouped or coarsened before a basis is computed and lifted back to ambient space. Implementations of 'lift()' for specific combinations (e.g., supervoxel + Slepian, parcel PCA) live in external packages or downstream code; fmrilatent ships only the contracts.

Slots

`mask` 'LogicalNeuroVol' defining the ambient domain.

`info` Optional list for implementation-specific metadata.

group_delta_loadings *Represent subject loadings as group shared loadings plus a delta*

Description

Represent subject loadings as group shared loadings plus a delta

Usage

```
group_delta_loadings(group, delta = NULL, scale = 1, id = NULL, meta = list())
```

Arguments

group	Matrix-like shared/group loadings or a 'SharedReference'.
delta	Optional subject-specific loading delta. If 'NULL', a zero delta is used.
scale	Numeric multiplier applied to 'delta'.
id	Optional id.
meta	Optional advisory metadata.

Value

A 'GroupDeltaLoadings' object.

haar_latent *Build Haar latent representation*

Description

Build Haar latent representation

Usage

```
haar_latent(
  X,
  mask,
  levels = NULL,
  z_seed = 42L,
  threshold = list(type = "none", value = 0)
)
```

Arguments

X	Numeric matrix (time x voxels within mask).
mask	LogicalNeuroVol or 3D logical array.
levels	Decomposition levels (optional).
z_seed	Integer Morton seed (default 42).
threshold	Threshold list passed to 'haar_wavelet_forward'.

Value

An object of class 'HaarLatent' containing coefficients and metadata.

`haar_meta` *Get metadata from Haar latent object*

Description

Get metadata from Haar latent object

Usage

```
haar_meta(x)
```

Arguments

`x`	A Haar latent object

Value

Metadata list or NULL

`haar_wavelet_forward` *Forward Haar wavelet transform (mask-adaptive, Morton order)*

Description

Forward Haar wavelet transform (mask-adaptive, Morton order)

Usage

```
haar_wavelet_forward(
  X,
  mask,
  levels = NULL,
  z_seed = 42L,
  threshold = list(type = "none", value = 0)
)
```

Arguments

`X`	Numeric matrix with time in rows and voxels (masked) in columns.
`mask`	3D logical array or LogicalNeuroVol defining the mask.
`levels`	Number of decomposition levels; defaults to ceil(log2(max(dim(mask)))).
`z_seed`	Integer seed for Morton ordering (tie-breaking).
`threshold`	List with fields 'type' ("none" | "absolute" | "relative_to_root_std") and 'value'.

Value

List with elements ‘coeff‘ (list(root, detail)), ‘meta‘ (counts, Morton hash, etc.).

`haar_wavelet_inverse` *Inverse Haar wavelet transform*

Description

Inverse Haar wavelet transform

Usage

```
haar_wavelet_inverse(
  coeff,
  mask,
  levels = NULL,
  z_seed = NULL,
  roi_mask = NULL,
  time_idx = NULL,
  levels_keep = NULL
)
```

Arguments

`coeff`	Output from ‘haar_wavelet_forward()‘ (list with root/detail matrices).
`mask`	3D logical array or LogicalNeuroVol.
`levels`	Integer levels (defaults to ‘meta$levels‘ if present).
`z_seed`	Integer Morton seed (defaults to ‘meta$z_seed‘).
`roi_mask`	Optional ROI mask (same dims as mask) to subset voxels before returning.
`time_idx`	Optional integer vector of time indices to subset rows.
`levels_keep`	Optional integer vector of detail levels to include (1 = finest, ‘levels‘ = coarsest). Levels not listed are zeroed, enabling coarse/partial reconstruction without full detail.

Value

Matrix (time x voxels) reconstructed (or subsetted if ROI/time specified).

heat_wavelet_latent *Heat wavelet latent constructor (explicit basis)*

Description

Heat wavelet latent constructor (explicit basis)

Usage

```
heat_wavelet_latent(
  X,
  mask,
  reduction = NULL,
  spec = basis_heat_wavelet(),
  k_neighbors = 6L,
  label = ""
)
```

Arguments

X	Matrix time x voxels (mask order)
mask	LogicalNeuroVol or 3D logical array
reduction	ClusterReduction; if NULL, defaults to one cluster per voxel
spec	heat wavelet spec (basis_heat_wavelet())
k_neighbors	k for local graph building
label	Optional label

Value

A ‘LatentNeuroVec‘ object.

heat_wavelet_loadings_handle *Construct a shared LoadingsHandle via heat-wavelet lifting*

Description

Wraps a heat-wavelet ‘lift()‘ call so multiple ‘LatentNeuroVec‘ instances can share the same spatial dictionary without embedding the full matrix in each object.

Usage

```
heat_wavelet_loadings_handle(
  reduction,
  basis_spec = basis_heat_wavelet(),
  data = NULL,
  k_neighbors = 6L,
  id = NULL,
  label = "heat-wavelet"
)
```

Arguments

`reduction`	Graph/cluster reduction used by 'lift()'.
`basis_spec`	Basis specification; defaults to 'basis_heat_wavelet()'.
`data`	Ignored by heat-wavelet lifting; accepted only to keep the lifted-handle constructor signature aligned across families.
`k_neighbors`	Number of neighbors used for local graph construction when materializing the lifted basis.
`id`	Optional registry id; provide a stable string to reuse across sessions. If NULL, a deterministic id is derived from the spec and reduction.
`label`	Optional human-readable label.

Value

A `LoadingsHandle`.

`HierarchicalBasisTemplate-class`

HierarchicalBasisTemplate Class

Description

A template-only, data-agnostic container for hierarchical Laplacian frames. Stores the sparse spatial dictionary (primal basis), a cached solver for the Gram matrix, and the parcellation hierarchy metadata. Intended to be built offline for a fixed template (e.g., MNI) and reused for encoding fMRI data.

Slots

`mask` LogicalNeuroVol defining the domain (3D).

`space` NeuroSpace (typically 4D with a singleton time dim) matching the mask.

`levels` List of integer vectors (one per level) giving parcel ids per voxel (mask order).

`parents` List mapping child parcel ids to parent ids for each level > 1.

`loadings` Sparse Matrix (voxels x atoms) containing concatenated atoms B.

`gram_factor` Cached factorization of $G = t(B)\%*\%B$ (e.g., dCHMsimpl from Matrix::Cholesky).

`atoms` data.frame describing each atom (col_id, level, parcel_id, parent_id, mode, label).

`meta` List for auxiliary metadata (atlas names, k_per_level, ridge, version).

`hrbf_generate_basis` *Hierarchical radial basis functions (HRBF) for latent fMRI*

Description

Utilities to generate analytic HRBF bases, project data, reconstruct, and build 'LatentNeuroVec' objects. Parameters are kept simple and in-R only (no descriptors or HDF5).

Usage

```
hrbf_generate_basis(params, mask)

hrbf_project_matrix(X, mask, params)

hrbf_reconstruct_matrix(coeff, mask, params)
```

Arguments

`params`	List with fields: - 'sigma0' (numeric, default 6) - 'levels' (integer, default 3) - 'radius_factor' (numeric, default 2.5) - 'num_extra_fine_levels' (integer, default 0) - 'kernel_type' (\"gaussian\", \"wendland_c4\", or \"wendland_c6\") - 'seed' (integer) for deterministic Poisson sampling
`mask`	'LogicalNeuroVol' mask defining voxel locations.
`X`	Numeric matrix with time in rows and voxels in columns.
`coeff`	Coefficient matrix with rows = time points.

Value

For 'hrbf_generate_basis', a sparse matrix with one row per HRBF atom and columns matching mask voxels.

`hrbf_latent` *Build a LatentNeuroVec using an HRBF basis*

Description

Build a LatentNeuroVec using an HRBF basis

Usage

```
hrbf_latent(X, mask, params = list(), label = "")
```

Arguments

`X`	Numeric matrix (time x voxels within mask).
`mask`	'LogicalNeuroVol'.
`params`	HRBF parameter list (see 'hrbf_generate_basis').
`label`	Optional character label.

Value

A 'LatentNeuroVec' with 'basis = coefficients', 'loadings = t(HRBF_basis)'.

`hrbf_meta` *Retrieve HRBF metadata if present*

Description

Retrieve HRBF metadata if present

Usage

```
hrbf_meta(x)
```

Arguments

`x`	A LatentNeuroVec object

Value

HRBF metadata list or NULL if not HRBF-tagged

hrbf_reconstruct_partial

Partially reconstruct selected voxels/timepoints

Description

Computes only the requested voxels (and optional timepoints) without materializing the full dense volume. Useful for fast indexed access.

Usage

```
hrbf_reconstruct_partial(coeff, mask, params, voxel_idx, time_idx = NULL)
```

Arguments

`coeff`	Coefficient matrix (time x atoms).
`mask`	'LogicalNeuroVol'.
`params`	HRBF parameter list.
`voxel_idx`	Integer vector of voxel linear indices (within the mask grid).
`time_idx`	Optional integer vector of time indices; defaults to all rows.

Value

Matrix of shape length(time_idx) x length(voxel_idx).

implicit_latent *Construct an ImplicitLatent object*

Description

Construct an ImplicitLatent object

Usage

```
implicit_latent(
  coeff,
  decoder,
  meta,
  mask = NULL,
  domain = NULL,
  support = NULL
)
```

Arguments

`coeff`	Arbitrary coefficient payload (list or matrix) needed by decoder.
`decoder`	Function(time_idx = NULL, roi_mask = NULL, levels_keep = NULL) returning matrix.
`meta`	List metadata; must include 'family' string.
`mask`	Logical 3D array (or LogicalNeuroVol) describing volumetric support.
`domain`	Optional decoded output domain. For non-volumetric latent objects, supply a domain such as a `neurosurf::SurfaceGeometry` together with `support`.
`support`	Optional decoded output support. For volumetric latent objects this is usually derived from `mask`; for surface-like latent objects it should be a vector of vertex indices (or a logical mask over vertices).

Value

An object of class 'ImplicitLatent'.

`implicit_meta` *Get metadata from ImplicitLatent object*

Description

Get metadata from ImplicitLatent object

Usage

```
implicit_meta(x)
```

Arguments

`x`	An ImplicitLatent object

Value

Metadata list or NULL

is_explicit_latent *Test whether a latent object is explicit*

Description

Test whether a latent object is explicit

Usage

```
is_explicit_latent(x, ...)

## S4 method for signature 'ExplicitLatent'
is_explicit_latent(x, ...)

## S4 method for signature 'ImplicitLatent'
is_explicit_latent(x, ...)
```

Arguments

x	A latent object.
...	Additional arguments (unused).

Value

Logical scalar; TRUE when the object stores explicit basis and loadings matrices.

is_haar_latent *Test if object is a Haar latent representation*

Description

Test if object is a Haar latent representation

Usage

```
is_haar_latent(x)
```

Arguments

x	Object to test

Value

Logical indicating if x is a Haar latent

is_hierarchical_template

Check whether an object is a HierarchicalBasisTemplate

Description

Check whether an object is a HierarchicalBasisTemplate

Usage

```
is_hierarchical_template(x)
```

Arguments

x	object to test

Value

Logical scalar.

is_hrbf_latent

Check if latent object carries HRBF metadata

Description

Check if latent object carries HRBF metadata

Usage

```
is_hrbf_latent(x)
```

Arguments

x	Object to test

Value

Logical indicating if x has HRBF metadata

is_implicit_latent *Test if object is an ImplicitLatent*

Description

Test if object is an ImplicitLatent

Usage

```
is_implicit_latent(x)
```

Arguments

x	Object to test

Value

Logical indicating if x is an ImplicitLatent

is_shared_reference *Test whether an object is a shared reference*

Description

Test whether an object is a shared reference

Usage

```
is_shared_reference(x)
```

Arguments

x	Object to test.

Value

Logical scalar.

is_surface_template *Test whether an object is a surface basis template*

Description

Test whether an object is a surface basis template

Usage

```
is_surface_template(x)
```

Arguments

x	Object to test.

Value

Logical scalar.

is_template *Test whether an object is a supported template*

Description

Test whether an object is a supported template

Usage

```
is_template(x)
```

Arguments

x	object to test

Value

Logical scalar.

is_transport_latent *Test whether an object is a transport-backed latent object*

Description

Test whether an object is a transport-backed latent object

Usage

```
is_transport_latent(x)
```

Arguments

x	Object to test.

Value

Logical scalar.

latent_dct_heatwavelet *Create a template LatentNeuroVec with heat-wavelet spatial loadings*

Description

Builds a `LatentNeuroVec` whose loadings are a heat-wavelet `LoadingsHandle` and whose basis is a placeholder zero matrix of `n_time x k` (where `k` is determined by the heat-wavelet loadings, not by the caller). The caller is expected to overwrite `lvec@basis` with real coefficients (e.g. fitted from data) before the object represents a valid factorization.

Usage

```
latent_dct_heatwavelet(
  n_time,
  k_time = NULL,
  mask,
  cluster_map = NULL,
  reduction = NULL,
  hw_basis_spec = NULL,
  offset = numeric(0),
  label = "DCT + heat-wavelet"
)
```

Arguments

n_time	Number of time points.
k_time	Optional ignored legacy argument. The number of components is determined by the heat-wavelet loadings. Supplying a non-NULL value warns with class 'fmrilatent_warning_deprecated'.
mask	LogicalNeuroVol or logical array mask (3D).
cluster_map	Optional integer vector mapping voxels (mask order) to clusters.
reduction	Graph reduction object; if NULL, built via 'make_cluster_reduction(mask, cluster_map)' with default one-cluster-per-voxel map.
hw_basis_spec	Heat-wavelet basis spec; defaults to 'basis_heat_wavelet()'.
offset	Optional voxel-wise offset (length n_vox).
label	Optional label.

Details

Despite the name, no DCT basis is constructed: the "dct" reference predates the spec/encoder pipeline and is retained for API compatibility. For an encoded DCT-temporal + heat-wavelet-spatial pipeline see `spec_st(time = spec_time_dct(...), space = spec_space_heat(...))` passed to `encode`.

Value

A `LatentNeuroVec` with placeholder basis matrix.

---

latent_domain *Extract the decoded domain associated with a latent object*

Description

Extract the decoded domain associated with a latent object

Usage

```
latent_domain(x, ...)

## S4 method for signature 'ImplicitLatent'
latent_domain(x, ...)

## S4 method for signature 'LatentNeuroSurfaceVector'
latent_domain(x, ...)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
latent_domain(x, ...)

## S4 method for signature 'BlockLatentNeuroVector'
```

```
latent_domain(x, ...)

## S4 method for signature 'LatentNeuroVec'
latent_domain(x, ...)
```

Arguments

x	A latent object.
...	Additional arguments passed to methods.

Value

Domain object or identifier.

---

latent_factory *Simple factory to build a spec and encode in one call*

Description

Simple factory to build a spec and encode in one call

Usage

```
latent_factory(
  family,
  x,
  mask,
  reduction = NULL,
  ...,
  materialize = "auto",
  label = ""
)
```

Arguments

family	Character scalar naming one of the standard ‘encode()‘ families. See **Accepted family names** for the canonical names and supported aliases.
x	Data matrix (time x voxels).
mask	Mask (required for spatial families).
reduction	Optional GraphReduction for spatial specs.
...	Passed to spec constructors and encode().
materialize	”handle”, ”matrix”, or ”auto” (default ”handle”).
label	Optional label for the resulting object.

Value

The class follows the same per-family contract as [encode()]: explicit spatial families and explicit temporal families return a [LatentNeuroVec] (a concrete 'ExplicitLatent'); the spatiotemporal families ('st_slepian', 'st_bspline_hrbf') build a 'spec_st' and therefore always return an 'ImplicitLatent'. See the '@return' section of [encode()] for the full taxonomy.

Accepted family names

Canonical names are listed first; aliases in parentheses are accepted for compatibility.

Temporal 'time_dct' ('dct_time'), 'time_slepian' ('slepian_time').

Spatial 'space_slepian' ('slepian_space'), 'space_pca' ('pca_space'), 'space_parcel' ('parcel_space'), 'space_heat' ('heat_space'), 'space_hrbf' ('hrbf_space'), 'space_wavelet_active' ('wavelet_active'), and 'hierarchical'.

Spatiotemporal 'st' (requires explicit 'time' and 'space' specs), 'st_slepian' ('slepian_st'), and 'st_bspline_hrbf' ('bspline_hrbf_st').

AWPT is intentionally not a 'latent_factory()' family because it requires a shared 'basis_asset' and a subject 'field_operator'; use [encode_awpt()] or [encode_operator()] for AWPT subject fitting.

`latent_meta` *Get lightweight metadata from a latent object*

Description

Get lightweight metadata from a latent object

Usage

```
latent_meta(x, ...)

## S4 method for signature 'ImplicitLatent'
latent_meta(x, ...)

## S4 method for signature 'LatentNeuroSurfaceVector'
latent_meta(x, ...)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
latent_meta(x, ...)

## S4 method for signature 'BlockLatentNeuroVector'
latent_meta(x, ...)

## S4 method for signature 'LatentNeuroVec'
latent_meta(x, ...)
```

Arguments

`x`	A latent object.
`...`	Additional arguments (unused).

Value

Metadata list.

`latent_searchlight` *Apply a user-defined function in latent space over neighborhoods*

Description

Runs a user-supplied function ‘fun‘ for each neighborhood using only latent quantities. ‘fun‘ is called with arguments ‘(B, L_V, M_V, idx, ...)‘, where: - ‘B‘ is the basis matrix (time x k) - ‘L_V‘ is the loadings restricted to the neighborhood (|V| x k) - ‘M_V = t(L_V) - ‘idx‘ is the voxel indices of the neighborhood ‘fun‘ should return any R object; results are collected in a list.

Usage

```
latent_searchlight(basis, loadings, neighborhoods, fun, ...)
```

Arguments

`basis`	Matrix or BasisHandle (time x k) from a LatentNeuroVec.
`loadings`	Matrix or LoadingsHandle (voxels x k) from a LatentNeuroVec.
`neighborhoods`	List of integer vectors of voxel indices (mask order).
`fun`	Function(B, L_V, M_V, idx, ...) returning a result per neighborhood.
`...`	Passed through to ‘fun‘.

Value

List of results, one per neighborhood.

`latent_support` *Extract the decoded support associated with a latent object*

Description

Extract the decoded support associated with a latent object

Usage

```
latent_support(x, ...)

## S4 method for signature 'ImplicitLatent'
latent_support(x, ...)

## S4 method for signature 'LatentNeuroSurfaceVector'
latent_support(x, ...)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
latent_support(x, ...)

## S4 method for signature 'BlockLatentNeuroVector'
latent_support(x, ...)

## S4 method for signature 'LatentNeuroVec'
latent_support(x, ...)
```

Arguments

`x`	A latent object.
`...`	Additional arguments passed to methods.

Value

Support object for the latent representation.

`LatentNeuroSurfaceVector` *Construct a LatentNeuroSurfaceVector*

Description

Construct a LatentNeuroSurfaceVector

Usage

```
LatentNeuroSurfaceVector(
  basis,
  loadings,
  geometry,
  support = NULL,
  offset = NULL,
  label = "",
  meta = list()
)
```

Arguments

basis	Temporal basis matrix (n_time x k) or BasisHandle.
loadings	Surface loadings matrix (n_support x k) or LoadingsHandle.
geometry	A neurosurf::SurfaceGeometry or neurosurf::SurfaceSet.
support	Surface support as vertex indices or a logical vector over all vertices.
offset	Optional numeric vector of length n_support.
label	Optional label.
meta	Optional metadata list.

Value

A LatentNeuroSurfaceVector.

LatentNeuroSurfaceVector-class

LatentNeuroSurfaceVector Class

Description

An explicit latent representation for surface-domain neuroimaging data. The data are stored as a temporal basis matrix and a surface loadings matrix over a supported set of vertices on a neurosurf geometry.

Slots

basis A Matrix or BasisHandle with dimensions (nTime x k).

loadings A Matrix or LoadingsHandle with dimensions (nVerticesInSupport x k).

offset Optional numeric vector of length nVerticesInSupport.

geometry Surface domain object, typically a neurosurf::SurfaceGeometry or neurosurf::SurfaceSet.

support Integer vector of vertex indices within geometry.

label Character label.

meta Lightweight metadata list.

`LatentNeuroVec` *Create a Latent Space Representation of Neuroimaging Data*

Description

Constructs a `LatentNeuroVec-class` object, which provides a memory-efficient representation of neuroimaging data using matrix factorization. This is particularly useful for dimensionality reduction techniques (e.g., PCA or ICA).

Usage

```
LatentNeuroVec(
  basis,
  loadings,
  space,
  mask,
  offset = NULL,
  label = "",
  meta = list(),
  expect_dense = FALSE
)
```

Arguments

`basis`	A numeric or `Matrix` object ($n \times k$) containing the temporal basis. Each row corresponds to a time point, each column to a component.
`loadings`	A numeric or `Matrix` object ($p \times k$) containing spatial loadings. Each row corresponds to a voxel within the mask, each column to a component.
`space`	A `NeuroSpace-class` defining the spatial/temporal dimensions. Must be 4-dimensional.
`mask`	A `LogicalNeuroVol-class` defining the brain mask. The number of TRUE values must equal nrow(loadings).
`offset`	Optional numeric vector of length p (voxel-wise offsets). If NULL, defaults to zero offset.
`label`	Optional character label for the object.
`meta`	Optional list of metadata (e.g., HRBF params or centres).
`expect_dense`	Logical; if 'TRUE', suppress the informational message emitted when a base-matrix 'basis'/'loadings' is dense (>50 families (e.g. diffusion wavelets) where a dense factor is by design. Defaults to 'FALSE', preserving the original message behavior.

Details

Construct a LatentNeuroVec Object

The data is represented as the product:

$$X = B \times L^T + c$$

where:

- B is the basis matrix ($n \times k$)
- L is the loadings matrix ($p \times k$)
- c is an optional offset vector (length p)
- n is the number of time points
- p is the number of voxels in the mask
- k is the number of components

Value

A new LatentNeuroVec-class instance.

Examples

```
# Example data
n_timepoints <- 4
n_components <- 2
mask_array <- array(TRUE, dim = c(2, 2, 1))
n_voxels <- sum(mask_array)

# Create basis & loadings
basis <- Matrix::Matrix(seq_len(n_timepoints * n_components),
  nrow = n_timepoints,
  ncol = n_components
)
loadings <- Matrix::Matrix(seq_len(n_voxels * n_components) / 10,
  nrow = n_voxels,
  ncol = n_components,
  sparse = TRUE
)

# Create space (2x2x1 volume, 4 timepoints)
spc <- neuroim2::NeuroSpace(c(2, 2, 1, n_timepoints))

# Create mask
mask_vol <- neuroim2::LogicalNeuroVol(mask_array, neuroim2::NeuroSpace(c(2, 2, 1)))

# Construct LatentNeuroVec
lvec <- LatentNeuroVec(
  basis = basis,
  loadings = loadings,
  space = spc,
  mask = mask_vol,
```

```
  expect_dense = TRUE
)
dim(lvec)
```

`LatentNeuroVec-class` *LatentNeuroVec Class*

Description

A class that represents a 4-dimensional neuroimaging array using a latent space decomposition. It stores the data as a set of basis functions (dictionary) and a corresponding set of loadings (coefficients), enabling efficient representation and manipulation of high-dimensional data.

Details

`LatentNeuroVec` inherits from `NeuroVec-class` and `AbstractSparseNeuroVec-class`. The original 4D data can be reconstructed as:

$$data[v, t] = \sum_k \big(basis[t, k] \times loadings[v, k]\big) + offset[v]$$

where v indexes voxels within the mask and t indexes time points.

This approach is especially useful for large datasets where storing the full 4D array is expensive. Common use cases include:

- PCA-reduced fMRI data
- ICA decompositions
- Dictionary learning representations
- Any matrix factorization of neuroimaging data

Slots

`basis` A `Matrix` or `BasisHandle` where each column represents a basis vector in the latent space. Dimensions are (nTime x k) where k is the number of components.

`loadings` A `Matrix` or `LoadingsHandle` (often sparse) containing the coefficients for each basis vector across the spatial dimensions. Dimensions are (nVoxels x k).

`offset` A `numeric` vector representing a constant offset term for each voxel or spatial location. Length is nVoxels (within mask).

`map` A `IndexLookupVol` object representing the mapping from 3D coordinates to linear indices within the mask.

`label` A `character` string representing the label for the latent vector.

`meta` A `list` storing lightweight metadata (e.g., HRBF params, centers).

Inheritance

`LatentNeuroVec` inherits from:

- `NeuroVec-class` - base 4D neuroimaging class
- `AbstractSparseNeuroVec-class` - sparse representation framework

See Also

`NeuroVec-class`, `AbstractSparseNeuroVec-class`, `LatentNeuroVec` constructor function.

Examples

```
n_timepoints <- 4
n_components <- 2
mask_array <- array(TRUE, dim = c(2, 2, 1))
n_voxels <- sum(mask_array)

# Create basis (temporal) & loadings (spatial)
basis <- Matrix::Matrix(seq_len(n_timepoints * n_components),
  nrow = n_timepoints, ncol = n_components
)
loadings <- Matrix::Matrix(seq_len(n_voxels * n_components) / 10,
  nrow = n_voxels, ncol = n_components, sparse = TRUE
)

# Create space (2x2x1 volume, 4 timepoints)
spc <- neuroim2::NeuroSpace(c(2, 2, 1, n_timepoints))

# Create mask
mask_vol <- neuroim2::LogicalNeuroVol(mask_array, neuroim2::NeuroSpace(c(2, 2, 1)))

# Construct LatentNeuroVec
lvec <- LatentNeuroVec(
  basis = basis,
  loadings = loadings,
  space = spc,
  mask = mask_vol,
  expect_dense = TRUE
)

dim(lvec)
```

`lift` *Lift reduced bases back to voxel space (abstract generic)*

Description

Lift reduced bases back to voxel space (abstract generic)

Usage

```
lift(reduction, basis_spec, data = NULL, ...)
```

Arguments

`reduction`	A 'GraphReduction' subclass describing topology.
`basis_spec`	A basis specification (e.g., 'basis_slepian()').
`data`	Optional data for data-driven bases (e.g., PCA).
`...`	Additional arguments passed to methods (e.g., k_neighbors).

Value

Typically a voxel x components 'Matrix' (often sparse) or an implementation-defined object for implicit decoders.

`lift,ClusterReduction,spec_diffusion_wavelet-method`

Lift diffusion wavelets for clustered reduction

Description

Lift diffusion wavelets for clustered reduction

Usage

```
## S4 method for signature 'ClusterReduction,spec_diffusion_wavelet'
lift(reduction, basis_spec, data = NULL, k_neighbors = 6L, ...)
```

Arguments

`reduction`	ClusterReduction describing voxel-to-cluster map.
`basis_spec`	Diffusion wavelet spec (basis_diffusion_wavelet()).
`data`	Ignored for this graph-only spatial dictionary; accepted only for the shared 'lift()' method signature.
`k_neighbors`	k for graph building (cluster centroids).
`...`	Additional arguments (unused).

Value

Matrix of loadings (voxels x components) concatenating scaling bases across scales.

`lift,ClusterReduction,spec_heat_wavelet-method`
Lift heat wavelets for clustered reduction

Description

Lift heat wavelets for clustered reduction

Usage

```
## S4 method for signature 'ClusterReduction,spec_heat_wavelet'
lift(reduction, basis_spec, data = NULL, k_neighbors = 6L, ...)
```

Arguments

`reduction`	ClusterReduction describing voxel-to-cluster map.
`basis_spec`	Heat wavelet spec (basis_heat_wavelet()).
`data`	Ignored for this graph-only spatial dictionary; accepted only for the shared 'lift()' method signature.
`k_neighbors`	k for local graph building.
`...`	Additional arguments (unused).

Value

Sparse Matrix (voxels x components).

`lift,ClusterReduction,spec_pca-method`
Lift parcel/cluster-local PCA bases for ClusterReduction

Description

Computes PCA eigenvectors within each cluster and assembles a global block-sparse loadings matrix (voxels x components). This is typically used with 'encode(..., spec_space_pca(...), reduction = ...)'.

Usage

```
## S4 method for signature 'ClusterReduction,spec_pca'
lift(
  reduction,
  basis_spec,
  data = NULL,
  center = TRUE,
  scale = FALSE,
```

```
  offset = NULL,
  backend = c("auto", "svds", "svd"),
  ...
)
```

Arguments

`reduction`	ClusterReduction describing voxel-to-cluster map.
`basis_spec`	PCA basis specification (from ‘basis_pca()‘).
`data`	Required numeric matrix (time x voxels in mask order). Unlike graph-only lift methods, PCA consumes ‘data‘ to estimate cluster-local components and aborts when it is ‘NULL‘.
`center`	Logical; center voxels before PCA (default TRUE).
`scale`	Logical; scale voxels before PCA (default FALSE).
`offset`	Optional numeric vector of voxel means (length n_vox). If provided and ‘center = TRUE‘, this is used instead of recomputing ‘colMeans(data)‘.
`backend`	SVD backend: ”auto” (default), ”svds” (RSpectra), or ”svd” (base).
`...`	Unused.

Details

‘lift(ClusterReduction, spec_pca)‘ always returns unwhitened spatial loadings. If ‘basis_spec$whiten‘ is ‘TRUE‘, the method emits a classed warning because whitening requires the projected temporal scores and is therefore the caller’s responsibility. The standard ‘encode(..., spec_space_pca(whiten = TRUE))‘ path handles this post-lift whitening using the ‘fmrilatent.singular_values‘ attribute attached here.

Value

A sparse Matrix (voxels x components) with attribute ‘fmrilatent.singular_values‘ giving per-component singular values.

`lift,ClusterReduction,spec_slepian-method`

Lift spatial Slepians for clustered reduction

Description

Lift spatial Slepians for clustered reduction

Usage

```
## S4 method for signature 'ClusterReduction,spec_slepian'
lift(reduction, basis_spec, data = NULL, k_neighbors = 6L, ...)
```

Arguments

`reduction`	ClusterReduction describing voxel-to-cluster map.
`basis_spec`	Slepian basis specification (from ‘basis_slepian()‘).
`data`	Ignored for this graph-only spatial dictionary; accepted only for the shared ‘lift()‘ method signature.
`k_neighbors`	k for local graph building.
`...`	Additional arguments (unused).

Value

Sparse Matrix (voxels x components), block-concatenated over clusters.

`lift,GraphReduction,ANY-method`

Default lift method (placeholder)

Description

This method exists to provide a clear error when no concrete lift is registered. External packages should implement methods for specific (reduction, basis_spec) signatures.

Usage

```
## S4 method for signature 'GraphReduction,ANY'
lift(reduction, basis_spec, data = NULL, ...)
```

Arguments

`reduction`	A ‘GraphReduction‘ subclass.
`basis_spec`	A basis specification object.
`data`	Optional data for data-driven bases.
`...`	Additional arguments (unused in default method).

Value

This method does not return; it aborts with a classed unsupported-operation error.

linear_access *Linear access to LatentNeuroVec elements*

Description

Access elements of a `LatentNeuroVec` using linear (1D) indexing. Reconstructs values on-the-fly from the basis and loadings factorization.

Access elements of a `LatentNeuroVec` using linear (1D) indices into the 4D array representation.

Usage

```
## S4 method for signature 'LatentNeuroVec,numeric'
linear_access(x, i)

## S4 method for signature 'LatentNeuroVec,integer'
linear_access(x, i)
```

Arguments

`x`	A `LatentNeuroVec` object
`i`	Numeric index vector

Value

Numeric vector of reconstructed values

The reconstructed values at the specified indices

list_encoders *List registered encoders*

Description

Returns a data.frame describing all registered encoder families, including both built-in encoders and those added by external packages.

Usage

```
list_encoders()
```

Value

A data.frame with columns: `family`, `description`, `package`.

Examples

```
list_encoders()
```

`lna_hrbf_basis_from_params` *Build an HRBF Basis with Optional Neuroarchive Compatibility Semantics*

Description

Compatibility entry point for external callers that need explicit control over centre/sigma inputs and output column-space semantics.

Usage

```
lna_hrbf_basis_from_params(
  params,
  mask,
  centres = NULL,
  sigmas = NULL,
  full_grid = TRUE,
  compat_profile = NULL,
  mask_world_coords = NULL,
  mask_arr = NULL,
  mask_linear_indices = NULL
)
```

Arguments

`params`	HRBF parameter list.
`mask`	'LogicalNeuroVol' mask defining voxel locations.
`centres`	Optional numeric matrix ('K x 3') of world-space centres.
`sigmas`	Optional numeric vector ('K') of sigma values for 'centres'.
`full_grid`	Logical. If 'TRUE', basis columns index the full mask grid ('length(as.array(mask))'); if 'FALSE', columns index active mask voxels.
`compat_profile`	Optional explicit compatibility profile identifier.
`mask_world_coords`	Optional precomputed active-voxel world coords.
`mask_arr`	Optional precomputed logical mask array.
`mask_linear_indices`	Optional precomputed active-voxel linear indices.

Value

Sparse matrix with one row per HRBF atom.

`load_hierarchical_template` *Load a hierarchical template from disk*

Description

Load a hierarchical template from disk

Usage

```
load_hierarchical_template(file)
```

Arguments

`file`	Path to .rds produced by save_hierarchical_template

Value

HierarchicalBasisTemplate

`load_template` *Load a saved template from disk*

Description

Load a saved template from disk

Usage

```
load_template(file)
```

Arguments

`file`	Path to a template RDS file.

Value

A supported template object.

`loadings` *Get the loadings matrix (spatial components)*

Description

Extract the loadings matrix from a latent space representation. For `LatentNeuroVec` objects, this returns the spatial loadings matrix with dimensions (nVoxels x k) where k is the number of components and nVoxels is the number of voxels within the mask.

Usage

```
loadings(x, ...)

## S4 method for signature 'LatentNeuroSurfaceVector'
loadings(x)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
loadings(x)

## S4 method for signature 'BlockLatentNeuroVector'
loadings(x)

## S4 method for signature 'LatentNeuroVec'
loadings(x)
```

Arguments

`x`	An object containing loadings (e.g., `LatentNeuroVec`)
`...`	Additional arguments (currently unused)

Value

The loadings matrix (typically voxels x components)

Examples

```
mask <- neuroim2::LogicalNeuroVol(
  array(TRUE, dim = c(2, 2, 1)),
  neuroim2::NeuroSpace(c(2, 2, 1))
)
lvec <- LatentNeuroVec(
  basis = matrix(1:6, nrow = 3),
  loadings = matrix(seq_len(8) / 10, nrow = 4),
  space = neuroim2::NeuroSpace(c(2, 2, 1, 3)),
  mask = mask,
  expect_dense = TRUE
)
l_matrix <- loadings(lvec)
```

```
dim(l_matrix)
```

`make_cluster_reduction` *Create a ClusterReduction from a mask and voxel-to-cluster map*

Description

Create a ClusterReduction from a mask and voxel-to-cluster map

Usage

```
make_cluster_reduction(mask, map)
```

Arguments

`mask`	A `LogicalNeuroVol` or logical 3D array defining the brain mask.
`map`	Integer vector (mask order) mapping each voxel to a cluster id.

Value

A `ClusterReduction` object.

`make_coarsened_reduction` *Create a coarsened graph reduction*

Description

Create a coarsened graph reduction

Usage

```
make_coarsened_reduction(mask, P_matrix, coarse_adj = NULL, info = list())
```

Arguments

`mask`	'LogicalNeuroVol' or array-like mask defining the fine domain.
`P_matrix`	Fine-by-coarse sparse prolongation matrix.
`coarse_adj`	Optional coarse-by-coarse sparse adjacency matrix.
`info`	Optional metadata list.

Value

A valid 'CoarsenedReduction'.

map *Get the map object*

Description

Extract the index map from a latent space representation. For `LatentNeuroVec` objects, this returns the `IndexLookupVol` that maps 3D coordinates to linear mask indices.

Usage

```
## S4 method for signature 'LatentNeuroVec'
map(x)
```

Arguments

`x`	An object containing a map (e.g., `LatentNeuroVec`)

Value

The index lookup object

mask *Get the mask*

Description

Extract the brain mask from a latent space representation. For `LatentNeuroVec` objects, this returns the `LogicalNeuroVol` defining which voxels are included in the representation.

Usage

```
## S4 method for signature 'ImplicitLatent'
mask(x)

## S4 method for signature 'LatentNeuroVec'
mask(x)
```

Arguments

`x`	An object containing a mask (e.g., `LatentNeuroVec`)

Value

The logical mask volume

mask_to_array *Convert mask to array*

Description

Safely converts a `LogicalNeuroVol` or array-like mask to a plain logical array, with informative error messages on failure.

Usage

```
mask_to_array(mask, location = "unknown function")
```

Arguments

`mask`	A `LogicalNeuroVol` or logical array.
`location`	Character string used in error messages to identify the caller.

Value

A logical array.

materialize_group_delta_loadings *Materialize group-plus-delta loadings*

Description

Materialize group-plus-delta loadings

Usage

```
materialize_group_delta_loadings(x)
```

Arguments

`x`	A ‘GroupDeltaLoadings‘ object.

Value

Matrix-like materialized loadings.

`materialize_shared_temporal_spec` *Materialize a shared temporal specification*

Description

Materialize a shared temporal specification

Usage

```
materialize_shared_temporal_spec(spec)
```

Arguments

`spec`	A 'SharedTemporalSpec'.

Value

Dense temporal basis matrix.

`neuroarchive_handoff_contract` *Create the fmrilatent-to-neuroarchive handoff contract*

Description

The handoff contract records the boundary between in-memory representation responsibilities owned by fmrilatent and persistent archive responsibilities owned by neuroarchive. It is intentionally a manifest, not a lazy archive reader or file locator.

Usage

```
neuroarchive_handoff_contract(
  representation = NULL,
  components = list(),
  templates = list(),
  references = list(),
  meta = list()
)
```

Arguments

`representation`	Optional fmrilatent representation object.
`components`	Optional list of shared component contracts.
`templates`	Optional list of reusable template assets.
`references`	Optional list of in-session shared references.
`meta`	Optional advisory metadata.

Value

A 'NeuroarchiveHandoffContract'.

`offset` *Get the offset vector*

Description

Extract the offset vector from a latent space representation. For `LatentNeuroVec` objects, this returns the voxel-wise offset (mean or intercept) that is added after the basis x loadings reconstruction.

Usage

```
## S4 method for signature 'ANY'
offset(object, ...)

## S4 method for signature 'LatentNeuroSurfaceVector'
offset(object, ...)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
offset(object, ...)

## S4 method for signature 'BlockLatentNeuroVector'
offset(object, ...)

## S4 method for signature 'LatentNeuroVec'
offset(object, ...)
```

Arguments

`object`	An object containing an offset (e.g., `LatentNeuroVec`)
`...`	Additional arguments for methods. The generic keeps the `object` first argument used by `stats::offset()` and adds `...` so offset accessors can follow the same extension pattern as sibling accessors such as `basis()` and `loadings()`.

Value

The offset vector (length = number of voxels in mask)

Examples

```
mask <- neuroim2::LogicalNeuroVol(
  array(TRUE, dim = c(2, 2, 1)),
  neuroim2::NeuroSpace(c(2, 2, 1))
)
```

```
lvec <- LatentNeuroVec(
  basis = matrix(1:6, nrow = 3),
  loadings = matrix(seq_len(8) / 10, nrow = 4),
  space = neuroim2::NeuroSpace(c(2, 2, 1, 3)),
  mask = mask,
  expect_dense = TRUE
)
off_vector <- offset(lvec)
length(off_vector)
```

`parcel_basis_template` *Build a shared parcel basis template*

Description

Computes a spatial dictionary within each parcel using `lift()` and caches the Gram factorization for efficient projection. The resulting template can be reused across subjects via `spec_space_parcel()`.

Usage

```
parcel_basis_template(
  parcellation,
  basis_spec = basis_slepian(k = 5),
  data = NULL,
  center = TRUE,
  ridge = 1e-08,
  ...
)
```

Arguments

`parcellation`	A `ClusterReduction` or a `ClusteredNeuroVol` (coerced via `as_cluster_reduction()`).
`basis_spec`	A basis specification for `lift()`. Default is `basis_slepian(k = 5)` which computes the k smallest Laplacian eigenvectors of the voxel adjacency graph within each parcel. Use `basis_pca(k)` with `data` for data-driven bases.
`data`	Optional numeric matrix (time x voxels, mask order). Required for data-driven specs like `basis_pca()`.
`center`	Logical; if `TRUE` (default), center voxel time series both when building PCA templates and when projecting new subjects. The encoded object stores subject-specific voxel means in `LatentNeuroVec@offset` so reconstruction preserves voxel means. For geometric bases (Laplacian, Slepian) this controls whether subjects are centered prior to projection.

`ridge` Small positive scalar added to the Gram diagonal if Cholesky fails (default 1e-8).

`...` Additional arguments passed to `lift()` (e.g., `k_neighbors` for graph-based specs). For PCA templates, projection must remain replayable at encode time, so preprocessing arguments such as `offset` and `scale` are not supported here.

Details

The default `basis_slepian(k)` computes the k smallest eigenvectors of the graph Laplacian built from voxel coordinates within each parcel. These smooth spatial functions form a data-independent dictionary suitable for projecting any subject's data.

For data-driven bases, pass `basis_pca(k)` with `data =` a training matrix (e.g., group-average data). The resulting PCA loadings are then fixed and reused for each subject.

Value

A `"ParcelBasisTemplate"` object (S3) with components:

loadings Sparse `Matrix` (voxels x atoms), block-diagonal by parcel.

gram_factor Cached Cholesky factorization of $L^T L$.

reduction `ClusterReduction` used.

basis_spec The spec that produced the loadings.

center Whether centering is applied when building/projecting.

meta List with `family`, `k`, `ridge`, `label_map`, and `cluster_map`.

See Also

`spec_space_parcel`, `as_cluster_reduction`, `lift`, `basis_slepian`, `basis_pca`

Examples

```
## Not run:
# Geometric (Laplacian) basis -- no training data needed
atlas <- load_atlas("schaefer_400")
tmpl <- parcel_basis_template(atlas, basis_slepian(k = 8))
lvec <- encode(bold, spec_space_parcel(tmpl))

# Data-driven shared PCA basis
tmpl_pca <- parcel_basis_template(atlas, basis_pca(k = 5), data = group_bold)
lvec <- encode(subj_bold, spec_space_parcel(tmpl_pca))

## End(Not run)
```

`parcel_similarity_matrix` *Build a similarity matrix for parcel clustering (Schaefer-like)*

Description

Combines geometric and functional priors into a parcel–parcel similarity matrix W used to cluster Schaefer parcels into coarser levels. Real data glue: - 'boundary_contact': from surface meshes (fsaverage), compute fraction of shared boundary between parcels. - 'geo_dist': geodesic distance between parcel centroids on the surface (fallback: Euclidean in MNI). - 'yeo17': network labels from the Schaefer/Yeo annotation (hemisphere-specific).

Usage

```
parcel_similarity_matrix(
  boundary_contact,
  geo_dist,
  yeo17,
  alpha = 0.5,
  beta = 0.3,
  gamma = 0.2,
  d0 = 30
)
```

Arguments

`boundary_contact`	Numeric matrix [n_parc x n_parc], symmetric, contact fraction (0..1).
`geo_dist`	Numeric matrix [n_parc x n_parc], symmetric geodesic distances (mm).
`yeo17`	Integer or factor vector of length n_parc with Yeo17 network labels.
`alpha`	Weight for boundary contact (default 0.5).
`beta`	Weight for geodesic kernel (default 0.3).
`gamma`	Weight for Yeo17 agreement (default 0.2).
`d0`	Scale for geodesic exponential (default 30 mm).

Value

Similarity matrix W (symmetric, zero diagonal).

parent_maps_from_levels

Derive parent maps for a nested set of parcellations

Description

Derive parent maps for a nested set of parcellations

Usage

```
parent_maps_from_levels(levels)
```

Arguments

`levels`	List of integer label vectors (all same length), coarse to fine or fine to coarse; order agnostic.

Value

List of integer named vectors: parents[[lvl]] maps child ids (names) at level lvl to parent ids at lvl-1; parents[[1]] is integer(0).

plot_basis_gram

Plot Gram matrix of a basis (orthogonality check)

Description

Plot Gram matrix of a basis (orthogonality check)

Usage

```
plot_basis_gram(basis)
```

Arguments

`basis`	Matrix or BasisHandle.

Value

ggplot heatmap (if ggplot2 available); otherwise shows base image.

`plot_benchmark_roundtrip` *Plot benchmark results*

Description

Plot benchmark results

Usage

```
plot_benchmark_roundtrip(df)
```

Arguments

`df`	Data frame from benchmark_roundtrip.

Value

ggplot object if ggplot2 available; otherwise prints df.

`plot_slepian_temporal` *Plot temporal Slepians (DPSS)*

Description

Plot temporal Slepians (DPSS)

Usage

```
plot_slepian_temporal(basis, max_components = 6L)
```

Arguments

`basis`	Matrix (time x k) or BasisHandle (kind = "slepian_temporal").
`max_components`	Maximum components to display (default 6).

Value

A ggplot object (if ggplot2 available); otherwise invisibly plots with base graphics.

plot_spatial_atom *Plot a spatial atom (loading vector) on a mask*

Description

Plot a spatial atom (loading vector) on a mask

Usage

```
plot_spatial_atom(loadings, mask, idx = 1L, main = NULL)
```

Arguments

loadings	Matrix (voxels x k) or LoadingsHandle.
mask	LogicalNeuroVol or logical array defining voxel order.
idx	Component index (1-based).
main	Optional title.

Value

Invisibly, the 3D array plotted.

portable_linear_map *Portable linear-map contract*

Description

The portable linear map is the seam 'fmrilatent' uses to consume subject field operators from neurofunctor (or any other producer) without pulling in file-format-specific warp code. A portable linear map represents a single linear operator $A : \mathbb{R}^{n_{\text{source}}} \to \mathbb{R}^{n_{\text{target}}}$ together with enough metadata to compose, adjoint, and (optionally) materialize it.

Details

An object satisfies the contract if it is either

1. a base `matrix` or `Matrix::Matrix`, or
2. a `list` with the following fields:

   `n_source, n_target` Positive integer scalar dimensions of the source and target sample vectors.

   `forward(x, ...)` Function applying the operator to source-space data. Accepts a vector of length `n_source` or a matrix with `n_source` rows; returns a vector/matrix with `n_target` rows.

   `adjoint_apply(y, ...)` **(alias** `adjoint`**)** Function applying the discrete adjoint (or a declared alternative) to target-space data.

`source_domain_id`, `target_domain_id` Stable character identifiers for the source and target domains. Used for composition safety and provenance digests. `.compose_linear_maps()` requires matching domain ids across the junction unless at least one side is an empty string ("unspecified").

`source_support`, `target_support` Optional descriptors of the sample layout on each side (for example a `LogicalNeuroVol`, a 3D logical mask, surface vertex indices, or a vector of sample ids). When present these let `fmrilatent` reconstruct domain-aware outputs; when absent it falls back to `provenance$target_support` / `provenance$target_mask`.

`adjoint_convention` Character tag identifying the adjoint convention. Defaults to `"euclidean_discrete"` (the discrete transpose under the standard Euclidean inner product). Covariance pushforward paths such as `decode_covariance()` and `.project_covariance_diag()` currently require this value; operators using another convention must be re-normalized before being handed to the pushforward code.

`provenance` Optional named list of provenance metadata. `fmrilatent` interprets a small set of reserved keys:

`target_mask`, `target_support`, `target_domain` Legacy fallback locations read by `.resolve_transport_target_support()` when the top-level fields are absent. New producers should set the top-level fields instead.

`source_support` Legacy fallback for the top-level field of the same name.

`coordinates` Appended with value `"raw"` when the map has been wrapped with `.transform_linear_map_coordinates()`.

All other keys are advisory and carried through composition verbatim. Producers should prefer top-level fields over provenance entries for anything fmrilatent actually reads.

`materialize(...)` **(optional)** Returns the dense `n_target` x `n_source` matrix representation. Used only when a caller explicitly asks for it; the main encode and decode paths are matrix-free and only call `forward`/`adjoint_apply`.

`contract_version` Integer scalar identifying the contract revision. Auto-filled by `as_portable_linear_map()` to `1L` in this release. Future revisions will bump this tag so producers and consumers can negotiate compatibility.

Producer-form vs canonical form on `TransportLatent`. `transport_latent()` records the raw producer object at `x$field_operator` (and its legacy alias `x$observation_operator`) as a back-reference for debugging and round-trip. The canonical portable-linear-map representation used internally by all encode/decode math is at `x$transport$field_operator`. Consumers that need to introspect the operator should always read the canonical form.

Producers (such as `neurofunctor::compile_observation_operator`) can return either a bare callback list or a wrapped object. Call `as_portable_linear_map()` to coerce any accepted input to the canonical contract and `validate_portable_linear_map()` to verify that an input already satisfies it without coercing.

predict.BoldZipSR *Predict from a BOLDZip-SR codec payload*

Description

Predict from a BOLDZip-SR codec payload

Usage

```
## S3 method for class 'BoldZipSR'
predict(object, time_idx = NULL, roi = NULL, ...)
```

Arguments

`object`	A 'BoldZipSR' object.
`time_idx`	Optional integer time indices to return.
`roi`	Optional integer or logical row subset to return.
`...`	Additional arguments passed to [boldzip_sr_decode()].

Value

Reconstructed matrix with rows as voxels/grayordinates and columns as time points.

predict.HaarLatent *Predict method for HaarLatent*

Description

Predict method for HaarLatent

Usage

```
## S3 method for class 'HaarLatent'
predict(object, roi_mask = NULL, time_idx = NULL, levels_keep = NULL, ...)
```

Arguments

`object`	HaarLatent object
`roi_mask`	Optional ROI mask
`time_idx`	Optional time indices
`levels_keep`	Optional levels to keep
`...`	Additional arguments passed to decoder

Value

Matrix of predicted values

`predict.ImplicitLatent` *Predict method for ImplicitLatent*

Description

Predict method for ImplicitLatent

Usage

```
## S3 method for class 'ImplicitLatent'
predict(object, roi_mask = NULL, time_idx = NULL, levels_keep = NULL, ...)
```

Arguments

`object`	ImplicitLatent object
`roi_mask`	Optional ROI mask
`time_idx`	Optional time indices
`levels_keep`	Optional levels to keep
`...`	Additional arguments (unused)

Value

Matrix of predicted values

`print.ParcelBasisTemplate` *Print method for ParcelBasisTemplate*

Description

Print method for ParcelBasisTemplate

Usage

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

Arguments

`x`	A ParcelBasisTemplate object.
`...`	Ignored.

Value

The input ‘x‘, invisibly.

print.SurfaceBasisTemplate *Print method for SurfaceBasisTemplate*

Description

Print method for SurfaceBasisTemplate

Usage

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

Arguments

x	A SurfaceBasisTemplate.
...	Ignored.

Value

The input 'x', invisibly.

project_effect *Compatibility wrapper for decoder-based coefficient projection*

Description

Compatibility wrapper for decoder-based coefficient projection

Usage

```
project_effect(
  x,
  gamma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  ...
)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
project_effect(
  x,
  gamma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  ...
```

```
)

## S4 method for signature 'ImplicitLatent'
project_effect(
  x,
  gamma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  ...
)

## S4 method for signature 'LatentNeuroSurfaceVector'
project_effect(
  x,
  gamma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  ...
)

## S4 method for signature 'LatentNeuroVec'
project_effect(
  x,
  gamma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  ...
)

## S4 method for signature 'BlockLatentNeuroVector'
project_effect(
  x,
  gamma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  ...
)
```

Arguments

`x`	A latent object.
`gamma`	Coefficient-space vector or matrix.
`space`	Output space to decode into.
`coordinates`	Coordinate system used by `gamma`.
`...`	Additional arguments passed to methods.

Value

Numeric vector or matrix in the requested output space.

---

project_hierarchical *Project coefficients only (no LatentNeuroVec wrapper)*

Description

Project coefficients only (no LatentNeuroVec wrapper)

Usage

```
project_hierarchical(template, X)
```

Arguments

`template`	HierarchicalBasisTemplate to use for projection
`X`	matrix time x voxels (mask order) matching template mask

Value

Matrix of coefficients (time x atoms)

---

project_vcov *Compatibility wrapper for decoder-based covariance pushforward*

Description

Compatibility wrapper for decoder-based covariance pushforward

Usage

```
project_vcov(
  x,
  Sigma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  diag_only = TRUE,
  ...
)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
project_vcov(
  x,
  Sigma,
```

```
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  diag_only = TRUE,
  ...
)

## S4 method for signature 'ImplicitLatent'
project_vcov(
  x,
  Sigma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  diag_only = TRUE,
  ...
)

## S4 method for signature 'LatentNeuroSurfaceVector'
project_vcov(
  x,
  Sigma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  diag_only = TRUE,
  ...
)

## S4 method for signature 'LatentNeuroVec'
project_vcov(
  x,
  Sigma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  diag_only = TRUE,
  ...
)

## S4 method for signature 'BlockLatentNeuroVector'
project_vcov(
  x,
  Sigma,
  space = c("native", "template"),
  coordinates = c("analysis", "raw"),
  diag_only = TRUE,
  ...
)
```

Arguments

x	A latent object.

`Sigma`	Coefficient covariance matrix.
`space`	Output space to decode into.
`coordinates`	Coordinate system used by `Sigma`.
`diag_only`	Logical; if `TRUE`, return only the diagonal.
`...`	Additional arguments passed to methods.

Value

Numeric vector or matrix in the requested output space.

`reconstruct_array` *Reconstruct a latent object as a 4D array*

Description

Reconstructs a latent object into a 4D array over its spatial support.

Usage

```
reconstruct_array(x, time_idx = NULL, roi_mask = NULL, ...)

## S4 method for signature 'ImplicitLatent'
reconstruct_array(x, time_idx = NULL, roi_mask = NULL, ...)

## S4 method for signature 'LatentNeuroSurfaceVector'
reconstruct_array(x, time_idx = NULL, roi_mask = NULL, ...)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
reconstruct_array(x, time_idx = NULL, roi_mask = NULL, ...)

## S4 method for signature 'BlockLatentNeuroVector'
reconstruct_array(x, time_idx = NULL, roi_mask = NULL, ...)

## S4 method for signature 'LatentNeuroVec'
reconstruct_array(x, time_idx = NULL, roi_mask = NULL, ...)
```

Arguments

`x`	A latent object.
`time_idx`	Optional integer time indices to keep.
`roi_mask`	Optional logical ROI mask; voxels outside the ROI are zero.
`...`	Additional arguments passed to methods.

Value

Numeric 4D array.

`reconstruct_matrix` *Reconstruct a latent object as a matrix*

Description

Provides a common reconstruction interface for both explicit `LatentNeuroVec` objects and implicit decoder-backed latent objects.

Usage

```
reconstruct_matrix(x, time_idx = NULL, roi_mask = NULL, ...)

## S4 method for signature 'ImplicitLatent'
reconstruct_matrix(x, time_idx = NULL, roi_mask = NULL, ...)

## S4 method for signature 'LatentNeuroSurfaceVector'
reconstruct_matrix(x, time_idx = NULL, roi_mask = NULL, ...)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
reconstruct_matrix(x, time_idx = NULL, roi_mask = NULL, ...)

## S4 method for signature 'BlockLatentNeuroVector'
reconstruct_matrix(x, time_idx = NULL, roi_mask = NULL, ...)

## S4 method for signature 'LatentNeuroVec'
reconstruct_matrix(x, time_idx = NULL, roi_mask = NULL, ...)
```

Arguments

`x`	A latent object.
`time_idx`	Optional integer time indices to keep.
`roi_mask`	Optional logical ROI mask for spatial subsetting.
`...`	Additional arguments passed to methods.

Value

Numeric matrix with rows = time and columns = voxels within the requested mask support.

`register_encoder` *Register an encoder family*

Description

Registers a spec constructor so it can be discovered via `list_encoders` and retrieved via `get_encoder`.

Usage

```
register_encoder(family, spec_fn, description = "", package = "")
```

Arguments

`family`	Character string identifying the encoder family (e.g. "time_slepian").
`spec_fn`	The spec constructor function (e.g. `spec_time_slepian`).
`description`	Character string describing the encoder.
`package`	Character string naming the package that provides this encoder.

Details

External packages typically register their encoders in their `.onLoad` hook so the registry is populated when the package is loaded:

```
.onLoad <- function(libname, pkgname) {
  if (requireNamespace("fmrilatent", quietly = TRUE)) {
    fmrilatent::register_encoder(
      "my_encoder", spec_my_encoder,
      description = "My custom encoder",
      package = pkgname
    )
  }
}
```

The S3 method itself is exported via the package's NAMESPACE in the usual way (`S3method(encode_spec, spec_my_encoder)`).

Registering the same family twice issues a warning and overwrites the previous entry.

Value

Invisibly, `TRUE`.

Dispatch model

The registry is for **discovery and introspection only**. It does *not* participate in actual encoding. `encode` routes work through the S3 generic `encode_spec`, which dispatches on the class of the spec object. That means an external package contributing a new encoder family must do **both**:

1. Define a spec constructor whose return value carries a distinctive class (e.g. `class(spec) <- c("spec_my_encoder", "list")`).
2. Define an S3 method `encode_spec.spec_my_encoder(x, spec, ...)` that builds the actual `LatentNeuroVec` (or other latent type).
3. Optionally call `register_encoder()` so that `list_encoders()` surfaces the family for users.

Calling `register_encoder()` alone is **not enough**: without an `encode_spec.spec_*` method, `encode()` will fall through to `encode_spec.default` and raise an error.

Transport-backed AWPT is an intentional parallel API, not a registry-dispatch family. `basis_awpt_wavelet()` describes a shared template basis, while subject fitting also needs a template/basis asset plus subject field or observation operators. Use `encode_awpt` or `encode_operator` for those fits; do not expect `register_encoder()` or `encode()` to route AWPT subject encoding.

See Also

`encode`, `list_encoders`, `get_encoder`

Examples

```
register_encoder("test_enc", identity, "A test encoder", "mypkg")
list_encoders()
get_encoder("test_enc")
```

`register_handle_kind` *Register a lazy handle materializer*

Description

Register a lazy handle materializer

Usage

```
register_handle_kind(kind, materializer, type = c("basis", "loadings"))
```

Arguments

`kind`	Character scalar handle kind.
`materializer`	Function that accepts a handle and returns a matrix-like object.
`type`	Registry to update: `"basis"` or `"loadings"`.

Value

Invisibly, the registered kind.

`render_shared_events` *Render sparse events with a shared event dictionary*

Description

Render sparse events with a shared event dictionary

Usage

```
render_shared_events(dictionary, events, n_atoms, n_time = NULL)
```

Arguments

`dictionary`	A 'SharedEventDictionary'.
`events`	Data frame with columns 'atom', 'time', 'amplitude', and optional 'shape_id'.
`n_atoms`	Number of event rows/atoms.
`n_time`	Number of time points. Defaults to the dictionary value.

Value

Sparse 'n_atoms x n_time' event matrix.

`resolve_shared_reference`
Resolve an in-session shared reference

Description

Resolve an in-session shared reference

Usage

```
resolve_shared_reference(x, cache = TRUE)
```

Arguments

`x`	A 'SharedReference', or any object returned unchanged.
`cache`	Cache materialized values in the session registry.

Value

The referenced object.

roi_subset_columns *Subset reconstruction matrix columns by ROI mask*

Description

Extracts columns from a reconstruction matrix that correspond to voxels inside an ROI mask. This is a shared utility used by decoder functions to handle ROI subsetting consistently.

Usage

```
roi_subset_columns(rec_mat, mask_arr, roi_mask = NULL)
```

Arguments

`rec_mat`	Numeric matrix (time x voxels-in-mask) to subset.
`mask_arr`	Logical array (the full brain mask).
`roi_mask`	Logical array (the ROI mask), or `NULL` to return `rec_mat` unchanged.

Value

The subsetted matrix, or `rec_mat` unchanged when `roi_mask` is `NULL`.

Examples

```
mask <- array(c(TRUE, TRUE, FALSE, TRUE), dim = c(2, 2, 1))
rec  <- matrix(1:9, nrow = 3, ncol = 3)  # 3 time x 3 masked voxels
roi  <- array(c(TRUE, FALSE, FALSE, TRUE), dim = c(2, 2, 1))
roi_subset_columns(rec, mask, roi)  # keeps columns 1 and 3
```

save_hierarchical_template *Save a hierarchical template to disk*

Description

Save a hierarchical template to disk

Usage

```
save_hierarchical_template(template, file, compress = "xz")
```

Arguments

`template`	HierarchicalBasisTemplate
`file`	Path to .rds file
`compress`	Compression passed to saveRDS (default ”xz”)

Value

The path in ‘file‘, invisibly.

`save_template` *Save a template object to disk*

Description

Save a template object to disk

Usage

```
save_template(template, file, compress = "xz", ...)

## S4 method for signature 'SurfaceAWPTBasisTemplate'
save_template(template, file, compress = "xz", ...)

## S4 method for signature 'AWPTBasisTemplate'
save_template(template, file, compress = "xz", ...)

## S4 method for signature 'HierarchicalBasisTemplate'
save_template(template, file, compress = "xz", ...)

## S4 method for signature 'ParcelBasisTemplate'
save_template(template, file, compress = "xz", ...)

## S4 method for signature 'SurfaceBasisTemplate'
save_template(template, file, compress = "xz", ...)
```

Arguments

`template`	A template object.
`file`	Output path.
`compress`	Compression passed to `saveRDS()`.
`...`	Additional arguments passed to methods.

Value

Normalized output path, invisibly.

`series` *Extract time series from LatentNeuroVec*

Description

Extract time series data from a `LatentNeuroVec`. Reconstructs values on-the-fly from the basis and loadings factorization.

Extract time series data from a `LatentNeuroVec` at specified spatial coordinates or voxel indices.

Usage

```
## S4 method for signature 'LatentNeuroVec,integer'
series(x, i, j, k, ..., drop = TRUE)

## S4 method for signature 'LatentNeuroVec,numeric'
series(x, i, j, k, ..., drop = TRUE)

## S4 method for signature 'LatentNeuroVec,ANY'
series(x, i, j, k, ..., drop = TRUE)
```

Arguments

`x`	A `LatentNeuroVec` object
`i`, `j`, `k`	Spatial indices (x, y, z coordinates)
`...`	Additional arguments passed to methods
`drop`	Logical; drop dimensions of length 1 (default TRUE)

Value

Matrix or vector of time series values

A matrix or vector of time series values

`shared_component_contract`
Build a method-neutral shared component contract

Description

A shared component contract describes a reusable spatial component matrix without committing to a particular decomposition method. The contract records dimensions, domain/support metadata, a digest, and optional advisory fields.

Usage

```
shared_component_contract(
  loadings,
  id = NULL,
  family = "shared_component",
  domain_id = "",
  support = NULL,
  measure = NULL,
  meta = list()
)
```

Arguments

loadings	Matrix-like component loadings (‘features x components‘), a ‘SharedReference‘ to such a matrix, or a ‘GroupDeltaLoadings‘ object.
id	Optional component id.
family	Method-neutral or method-family label.
domain_id	Stable domain identifier.
support	Optional support descriptor whose cardinality must match ‘nrow(loadings)‘ when supplied.
measure	Optional support measure. Vectors must have one value per feature; matrices must be square over features.
meta	Optional advisory metadata.

Value

A ‘SharedComponentContract‘ object.

---

shared_event_dictionary

Construct a reusable event shape dictionary

Description

Construct a reusable event shape dictionary

Usage

```
shared_event_dictionary(
  shapes = list(impulse = 1),
  n_time = NULL,
  meta = list()
)
```

Arguments

`shapes`	Named list of numeric shape vectors.
`n_time`	Optional temporal length used for validation.
`meta`	Optional advisory metadata.

Value

A 'SharedEventDictionary'.

`shared_reference` *Create an in-session shared object reference*

Description

'shared_reference()' records a reusable object behind a stable in-session identifier. It deliberately does not create files, archive locators, HDF5 datasets, checksums, or persistent registries; those responsibilities belong to archive packages such as neuroarchive.

Usage

```
shared_reference(
  value = NULL,
  id = NULL,
  kind = "object",
  materialize = NULL,
  meta = list(),
  register = TRUE
)
```

Arguments

`value`	Optional object to cache immediately.
`id`	Optional reference id. If omitted, a deterministic digest is used when 'value' is present; otherwise a unique session id is generated.
`kind`	Human-readable object kind.
`materialize`	Optional zero-argument function used to create the value on first resolution.
`meta`	Optional advisory metadata.
`register`	If 'TRUE', cache 'value' in the session registry.

Value

A 'SharedReference' descriptor.

`shared_reference_clear` *Reset the in-session shared reference cache*

Description

Reset the in-session shared reference cache

Usage

```
shared_reference_clear()
```

Value

Invisibly, ‘TRUE‘.

`shared_reference_info` *Summarize a shared reference without resolving it*

Description

Summarize a shared reference without resolving it

Usage

```
shared_reference_info(x)
```

Arguments

`x`	A ‘SharedReference‘.

Value

A list with id, kind, cache state, persistence, and metadata.

shared_temporal_spec *Construct a shared temporal basis descriptor*

Description

Construct a shared temporal basis descriptor

Usage

```
shared_temporal_spec(
  kind = c("dct", "bspline", "slepian", "custom"),
  n_time = NULL,
  rank = NULL,
  basis = NULL,
  params = list(),
  id = NULL,
  meta = list()
)
```

Arguments

kind	Temporal basis kind: '"dct"', '"bspline"', '"slepian"', or '"custom"'.
n_time	Number of time points. Inferred from 'basis' for custom specs.
rank	Number of basis columns. Inferred from 'basis' when omitted.
basis	Matrix-like basis for 'kind = "custom"'.
params	Basis-specific parameters.
id	Optional id.
meta	Optional advisory metadata.

Value

A 'SharedTemporalSpec' descriptor.

show *Display a LatentNeuroVec object*

Description

Print a summary of a `LatentNeuroVec` object including dimensions, component count, memory usage, and sparsity information.

Print a formatted summary of a `LatentNeuroVec` object showing dimensions, components, memory usage, and other relevant information.

Usage

```
## S4 method for signature 'LatentNeuroSurfaceVector'
show(object)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
show(object)

## S4 method for signature 'BlockLatentNeuroVector'
show(object)

## S4 method for signature 'LatentNeuroVec'
show(object)
```

Arguments

`object`	A `LatentNeuroVec` object

Value

Called for side effect (printing). Returns `NULL` invisibly.

Invisibly returns NULL; called for side effect of printing

`slepian_spatial_latent` *Slepian spatial latent constructor (explicit basis)*

Description

Slepian spatial latent constructor (explicit basis)

Usage

```
slepian_spatial_latent(
  X,
  mask,
  reduction = NULL,
  spec = basis_slepian(),
  k_neighbors = 6L,
  label = ""
)
```

Arguments

`X`	Matrix time x voxels (mask order).
`mask`	LogicalNeuroVol or 3D logical array.
`reduction`	ClusterReduction; if NULL, defaults to one cluster per voxel.

spec	Slepian basis spec (basis_slepian()).
k_neighbors	k for local graph building.
label	Optional label.

Value

A 'LatentNeuroVec' object.

`slepian_spatial_loadings_handle` *Create a LoadingsHandle for spatial Slepians (graph Laplacian)*

Description

Create a LoadingsHandle for spatial Slepians (graph Laplacian)

Usage

```
slepian_spatial_loadings_handle(
  reduction,
  basis_spec = basis_slepian(),
  data = NULL,
  k_neighbors = 6L,
  id = NULL,
  label = "slepian-spatial"
)
```

Arguments

reduction	Graph reduction (e.g., ClusterReduction).
basis_spec	Slepian basis spec (from 'basis_slepian()').
data	Optional data passed to 'lift()' (if needed).
k_neighbors	Number of neighbors used for local graph construction when materializing the lifted basis.
id	Optional registry id; generated if NULL.
label	Optional label.

Details

This constructor lifts the spatial dictionary eagerly so the returned handle records the realized dimensions and registers a fingerprinted cache entry. Repeated constructor calls may therefore recompute the lift even when later 'loadings_mat()' calls can reuse the registry cache.

Value

A `LoadingsHandle`.

`slepian_spatiotemporal_latent` *Spatiotemporal Slepian latent (implicit, separable)*

Description

Spatiotemporal Slepian latent (implicit, separable)

Usage

```
slepian_spatiotemporal_latent(
  X,
  mask,
  tr,
  bandwidth = 0.1,
  k_time = NULL,
  reduction = NULL,
  k_space = 3L,
  k_neighbors = 6L,
  label = ""
)
```

Arguments

`X`	Numeric matrix time x voxels (mask order).
`mask`	LogicalNeuroVol or 3D logical array.
`tr`	Repetition time (seconds).
`bandwidth`	Half-bandwidth in Hz for temporal Slepians (default 0.1).
`k_time`	Number of temporal Slepians; if NULL uses floor(2*NW)-1.
`reduction`	ClusterReduction for spatial graph; if NULL, one cluster per voxel.
`k_space`	Number of spatial Slepians per cluster (default 3).
`k_neighbors`	k-NN for spatial graph (default 6).
`label`	Optional label.

Value

An ‘ImplicitLatent‘ with decoder using separable Slepians.

`slepian_temporal_handle`
Create a BasisHandle for temporal Slepians (DPSS)

Description

Create a BasisHandle for temporal Slepians (DPSS)

Usage

```
slepian_temporal_handle(
  n_time,
  tr,
  bandwidth = 0.1,
  k = NULL,
  backend = c("tridiag", "dense"),
  id = NULL,
  label = NULL
)
```

Arguments

`n_time`	Integer number of time points.
`tr`	Repetition time (seconds).
`bandwidth`	Half-bandwidth in Hz (default 0.1).
`k`	Optional number of tapers/components.
`backend`	Backend passed to ‘dpss_time_basis‘; only ”tridiag” is currently supported.
`id`	Optional registry key (generated if NULL).
`label`	Optional human-readable label.

Value

A `BasisHandle`.

`slepian_temporal_latent`
LatentNeuroVec using a temporal DPSS basis

Description

LatentNeuroVec using a temporal DPSS basis

Usage

```
slepian_temporal_latent(
  X,
  mask,
  tr,
  bandwidth = 0.1,
  k = NULL,
  denoise = TRUE,
  backend = c("tridiag", "dense"),
  label = ""
)
```

Arguments

`X`	Numeric matrix (time x voxels within mask).
`mask`	'LogicalNeuroVol'.
`tr`	Repetition time in seconds.
`bandwidth`	Half-bandwidth in Hz (default 0.1).
`k`	Optional number of tapers; see 'dpss_time_basis'.
`denoise`	If TRUE, truncate to 'floor(2 * NW) - 1' (Shannon number).
`backend`	DPSS computation backend passed to 'dpss_time_basis'; only `"tridiag"` is currently supported.
`label`	Optional character label.

Value

'LatentNeuroVec' with DPSS temporal basis and voxel loadings.

`spec_hierarchical_template` *Create hierarchical template spec*

Description

Create hierarchical template spec

Usage

```
spec_hierarchical_template(template = NULL, template_file = NULL)
```

Arguments

`template`	HierarchicalBasisTemplate object
`template_file`	Path to saved template file

Value

Spec object of class spec_hierarchical

`spec_space_heat` *Spatial heat-wavelet spec (graph diffusion)*

Description

Spatial heat-wavelet spec (graph diffusion)

Usage

```
spec_space_heat(
  scales = c(1, 2, 4, 8),
  order = 30L,
  threshold = NULL,
  k_neighbors = 6L,
  sparsify_eps = NULL
)
```

Arguments

`scales`	Heat scales.
`order`	Polynomial order.
`threshold`	Deprecated alias for ‘sparsify_eps‘.
`k_neighbors`	k-NN graph parameter.
`sparsify_eps`	Non-negative threshold for small heat-wavelet coefficients. This is stored as ‘threshold‘ for compatibility.

Value

A ‘spec_space_heat‘ object.

`spec_space_hrbf` *Spatial HRBF spec*

Description

Spatial HRBF spec

Usage

```
spec_space_hrbf(
  params = list(),
  sigma0 = NULL,
  levels = NULL,
  radius_factor = NULL,
  num_extra_fine_levels = NULL,
  kernel_type = NULL,
  kernel_type_fine_levels = NULL,
  seed = NULL,
  ...
)
```

Arguments

`params`	Optional named list for advanced or compatibility HRBF parameters. Explicit formals override entries with the same names.
`sigma0`	Base kernel width. 'NULL' uses the HRBF default ('6').
`levels`	Number of dyadic resolution levels. 'NULL' uses the HRBF default ('3L').
`radius_factor`	Atom support radius as a multiple of level width. 'NULL' uses the HRBF default ('2.5').
`num_extra_fine_levels`	Optional extra fine levels. 'NULL' uses the HRBF default ('0L').
`kernel_type`	Radial kernel family. 'NULL' uses '"gaussian"'.
`kernel_type_fine_levels`	Kernel family for extra fine levels. 'NULL' uses the HRBF default.
`seed`	Random seed. 'NULL' uses the HRBF default ('1L').
`...`	Additional named HRBF parameters for advanced use.

Value

A 'spec_space_hrbf' object.

`spec_space_parcel` *Spatial parcel-basis spec (shared/template-based)*

Description

Creates a spec for projecting data onto a pre-built `"ParcelBasisTemplate"`. The loadings are fixed (shared across subjects); only the temporal scores and per-subject offset vary.

Usage

```
spec_space_parcel(template)
```

Arguments

template	A "ParcelBasisTemplate" object built by parcel_basis_template().

Value

A spec_space_parcel object for encode().

See Also

parcel_basis_template, encode

Examples

```
## Not run:
tmpl <- parcel_basis_template(atlas, basis_slepian(k = 8))
lvec_s1 <- encode(bold_s1, spec_space_parcel(tmpl))
lvec_s2 <- encode(bold_s2, spec_space_parcel(tmpl))
# Same loadings; different basis (scores) and offset

## End(Not run)
```

---

spec_space_pca *Spatial PCA spec (cluster-local)*

Description

Computes PCA eigenvectors within each cluster/parcel specified by a ‘ClusterReduction‘ and returns a block-sparse spatial dictionary.

Usage

```
spec_space_pca(
  k = NULL,
  center = TRUE,
  whiten = FALSE,
  backend = c("auto", "svds", "svd"),
  scale = FALSE
)
```

Arguments

k	Optional components per cluster. If ‘NULL‘, encoders use their family default (‘3L‘).
center	Logical; center voxels before PCA (default TRUE). When TRUE, voxel means are stored in ‘LatentNeuroVec@offset‘.
whiten	Logical; if TRUE, return whitened scores (unit-variance) and rescaled loadings such that reconstruction is unchanged.

backend	SVD backend: "auto" (default), "svds" (RSpectra), or "svd" (base).
scale	Logical; scale voxels before PCA (default FALSE). Passed through to 'lift(ClusterReduction, spec_pca)'.

Value

A 'spec_space_pca' object.

---

spec_space_slepian *Spatial Slepian spec*

Description

Spatial Slepian spec

Usage

```
spec_space_slepian(k = NULL, k_neighbors = 6L)
```

Arguments

k	Optional components per cluster. If 'NULL', encoders use their family default ('3L').
k_neighbors	k-NN graph parameter.

Value

A 'spec_space_slepian' object.

---

spec_space_wavelet_active *Spatial wavelet (active pencil) spec*

Description

Spatial wavelet (active pencil) spec

Usage

```
spec_space_wavelet_active(
  levels_space = 2L,
  levels_time = 0L,
  select_threshold = NULL,
  threshold = NULL
)
```

Arguments

`levels_space`	Spatial lifting levels.
`levels_time`	Optional time lifting levels.
`select_threshold`	Non-negative active-set coefficient threshold after the spatial transform. This is stored as 'threshold' for compatibility.
`threshold`	Deprecated alias for 'select_threshold'.

Value

A 'spec_space_wavelet_active' object.

`spec_st` *Spatiotemporal spec (separable)*

Description

Spatiotemporal spec (separable)

Usage

```
spec_st(time, space, core_mode = c("auto", "explicit"))
```

Arguments

`time`	Temporal spec ('spec_time_*').
`space`	Spatial spec ('spec_space_*').
`core_mode`	Controls the returned representation. '"auto"' (the default) always returns an 'ImplicitLatent' (decoder-only separable form). '"explicit"' returns an explicit [LatentNeuroVec] (the separable 'basis materializable, which holds for every currently supported 'spec_st' combination; if the factors are not both materializable it falls back to 'ImplicitLatent' with a 'fmrilatent_warning_explicit_core_unavailable' warning. The reconstruction is identical either way.

Value

A 'spec_st' object.

spec_time_bspline *Temporal B-spline spec*

Description

Temporal B-spline spec

Usage

```
spec_time_bspline(
  k = NULL,
  degree = 3L,
  include_intercept = FALSE,
  orthonormalize = TRUE
)
```

Arguments

k	Optional number of components (df). If 'NULL', the encoder uses 'min(5, n_time)' at encode time.
degree	Spline degree (default 3).
include_intercept	Logical include intercept.
orthonormalize	Logical orthonormalize columns (default TRUE).

Value

A 'spec_time_bspline' object.

spec_time_dct *Temporal DCT spec*

Description

Temporal DCT spec

Usage

```
spec_time_dct(k = NULL, norm = c("ortho", "none"))
```

Arguments

k	Optional number of components. If 'NULL', the encoder uses all available DCT columns ('k = n_time') at encode time.
norm	Normalization ("ortho" or "none").

Value

A ‘spec_time_dct‘ object.

`spec_time_slepian` *Temporal Slepian/DPSS spec*

Description

Temporal Slepian/DPSS spec

Usage

```
spec_time_slepian(
  tr,
  bandwidth = 0.1,
  k = NULL,
  backend = c("tridiag", "dense")
)
```

Arguments

`tr`	Repetition time (seconds).
`bandwidth`	Half-bandwidth in Hz (default 0.1).
`k`	Optional number of components (default floor(2*NW)-1).
`backend`	Backend to use. Only ”tridiag” is currently supported.

Value

A ‘spec_time_slepian‘ object for ‘encode()‘ / ‘spec_st()‘.

`spectral_ward_hclust` *Run spectral+Ward hierarchical clustering on a parcel graph*

Description

Produces an hclust over parcels using a spectral embedding of the similarity matrix, then Ward linkage. Real data glue: supply W from ‘parcel_similarity_matrix()‘, ‘hemi‘ from parcel hemisphere labels, and ‘network‘ from Yeo17 labels. Penalties bias merges to stay within hemisphere/network until higher in the tree.

Usage

```
spectral_ward_hclust(W, k_embed = 3, hemi = NULL, network = NULL)
```

Arguments

W	Similarity matrix (symmetric).
k_embed	Embedding dimension (2 or 3).
hemi	Optional factor/character vector (n_parc) to enforce within-hemisphere merges first.
network	Optional factor/character vector (n_parc) (e.g., Yeo17) to enforce within-network merges until relaxed.

Value

hclust object.

---

surface_basis_template

Build a shared surface basis template

Description

Build a shared surface basis template

Usage

```
surface_basis_template(
  geometry,
  loadings,
  support = NULL,
  roughness = NULL,
  measure = NULL,
  ridge = 1e-08,
  label = "surface_basis",
  meta = list()
)
```

Arguments

geometry	A `neurosurf::SurfaceGeometry` or `neurosurf::SurfaceSet`.
loadings	A matrix-like decoder basis with rows aligned to `support` and columns aligned to latent coefficients.
support	Surface support as vertex indices or a logical vector over all surface nodes. Defaults to the full surface.
roughness	Optional coefficient-space roughness matrix.
measure	Optional support-aligned weighting or mass information.
ridge	Small diagonal ridge added to the Gram matrix if needed.
label	Optional label stored in metadata.
meta	Optional additional metadata.

Value

A `SurfaceBasisTemplate` object.

`template_domain` *Extract the domain associated with a template*

Description

Extract the domain associated with a template

Usage

```
template_domain(template, ...)

## S4 method for signature 'SurfaceAWPTBasisTemplate'
template_domain(template, ...)

## S4 method for signature 'AWPTBasisTemplate'
template_domain(template, ...)

## S4 method for signature 'HierarchicalBasisTemplate'
template_domain(template, ...)

## S4 method for signature 'ParcelBasisTemplate'
template_domain(template, ...)

## S4 method for signature 'SurfaceBasisTemplate'
template_domain(template, ...)
```

Arguments

`template`	A template object.
`...`	Additional arguments passed to methods.

Value

Domain object or identifier.

template_loadings *Extract template loadings*

Description

Extract template loadings

Usage

```
template_loadings(x, ...)

## S4 method for signature 'SurfaceAWPTBasisTemplate'
template_loadings(x, ...)

## S4 method for signature 'AWPTBasisTemplate'
template_loadings(x, ...)

## S4 method for signature 'HierarchicalBasisTemplate'
template_loadings(x, ...)

## S4 method for signature 'ParcelBasisTemplate'
template_loadings(x, ...)

## S4 method for signature 'SurfaceBasisTemplate'
template_loadings(x, ...)
```

Arguments

x	A template object.
...	Additional arguments (unused).

Value

Spatial loadings matrix stored by the template.

template_mask *Extract template mask*

Description

Extract template mask

Usage

```
template_mask(x, ...)

## S4 method for signature 'SurfaceAWPTBasisTemplate'
template_mask(x, ...)

## S4 method for signature 'AWPTBasisTemplate'
template_mask(x, ...)

## S4 method for signature 'HierarchicalBasisTemplate'
template_mask(x, ...)

## S4 method for signature 'ParcelBasisTemplate'
template_mask(x, ...)

## S4 method for signature 'SurfaceBasisTemplate'
template_mask(x, ...)
```

Arguments

x	A template object.
...	Additional arguments (unused).

Value

Mask associated with the template.

---

template_measure *Extract optional measure or mass information for a template*

Description

Extract optional measure or mass information for a template

Usage

```
template_measure(template, ...)

## S4 method for signature 'SurfaceAWPTBasisTemplate'
template_measure(template, ...)

## S4 method for signature 'AWPTBasisTemplate'
template_measure(template, ...)

## S4 method for signature 'HierarchicalBasisTemplate'
template_measure(template, ...)
```

```
## S4 method for signature 'ParcelBasisTemplate'
template_measure(template, ...)

## S4 method for signature 'SurfaceBasisTemplate'
template_measure(template, ...)
```

Arguments

`template`	A template object.
`...`	Additional arguments passed to methods.

Value

Optional measure or mass object, or `NULL`.

`template_meta` *Extract template metadata*

Description

Extract template metadata

Usage

```
template_meta(x, ...)

## S4 method for signature 'SurfaceAWPTBasisTemplate'
template_meta(x, ...)

## S4 method for signature 'AWPTBasisTemplate'
template_meta(x, ...)

## S4 method for signature 'HierarchicalBasisTemplate'
template_meta(x, ...)

## S4 method for signature 'ParcelBasisTemplate'
template_meta(x, ...)

## S4 method for signature 'SurfaceBasisTemplate'
template_meta(x, ...)
```

Arguments

`x`	A template object.
`...`	Additional arguments (unused).

Value

Metadata list.

template_project *Project data onto a template*

Description

Project data onto a template

Usage

```
template_project(x, data, ...)

## S4 method for signature 'SurfaceAWPTBasisTemplate'
template_project(x, data, ...)

## S4 method for signature 'AWPTBasisTemplate'
template_project(x, data, ...)

## S4 method for signature 'HierarchicalBasisTemplate'
template_project(x, data, ...)

## S4 method for signature 'ParcelBasisTemplate'
template_project(x, data, ...)

## S4 method for signature 'SurfaceBasisTemplate'
template_project(x, data, ...)
```

Arguments

`x`	A template object.
`data`	Numeric matrix (time x voxels in mask order).
`...`	Additional arguments passed to methods.

Value

A list with `coefficients` (time x atoms/components) and `offset` (voxel means or `numeric(0)`).

template_rank *Query the rank of a template basis*

Description

Query the rank of a template basis

Usage

```
template_rank(template, ...)

## S4 method for signature 'SurfaceAWPTBasisTemplate'
template_rank(template, ...)

## S4 method for signature 'AWPTBasisTemplate'
template_rank(template, ...)

## S4 method for signature 'HierarchicalBasisTemplate'
template_rank(template, ...)

## S4 method for signature 'ParcelBasisTemplate'
template_rank(template, ...)

## S4 method for signature 'SurfaceBasisTemplate'
template_rank(template, ...)
```

Arguments

`template`	A template object.
`...`	Additional arguments passed to methods.

Value

Integer scalar rank.

`template_roughness` *Extract the spatial roughness operator for a template asset*

Description

Extract the spatial roughness operator for a template asset

Usage

```
template_roughness(template, coordinates = c("analysis", "raw"), ...)

## S4 method for signature 'SurfaceAWPTBasisTemplate'
template_roughness(template, coordinates = c("analysis", "raw"), ...)

## S4 method for signature 'AWPTBasisTemplate'
template_roughness(template, coordinates = c("analysis", "raw"), ...)

## S4 method for signature 'HierarchicalBasisTemplate'
template_roughness(template, coordinates = c("analysis", "raw"), ...)
```

```
## S4 method for signature 'ParcelBasisTemplate'
template_roughness(template, coordinates = c("analysis", "raw"), ...)

## S4 method for signature 'SurfaceBasisTemplate'
template_roughness(template, coordinates = c("analysis", "raw"), ...)
```

Arguments

`template`	A shared basis asset.
`coordinates`	Coordinate system for the returned roughness operator.
`...`	Additional arguments passed to methods.

Value

Matrix-like roughness operator or `NULL`.

`template_support` *Extract the support associated with a template*

Description

Extract the support associated with a template

Usage

```
template_support(template, ...)

## S4 method for signature 'SurfaceAWPTBasisTemplate'
template_support(template, ...)

## S4 method for signature 'AWPTBasisTemplate'
template_support(template, ...)

## S4 method for signature 'HierarchicalBasisTemplate'
template_support(template, ...)

## S4 method for signature 'ParcelBasisTemplate'
template_support(template, ...)

## S4 method for signature 'SurfaceBasisTemplate'
template_support(template, ...)
```

Arguments

`template`	A template object.
`...`	Additional arguments passed to methods.

Value

Support object describing the decoded support of the template.

`transport_latent` *Construct a transport-backed implicit latent object*

Description

Construct a transport-backed implicit latent object

Usage

```
transport_latent(
  coeff_raw,
  basis_asset,
  field_operator = NULL,
  observation_operator = NULL,
  mask = NULL,
  domain = NULL,
  support = NULL,
  coeff_analysis = NULL,
  analysis_transform = NULL,
  offset = numeric(0),
  run_info = NULL,
  meta = list()
)
```

Arguments

`coeff_raw`	Raw coefficient matrix (time x k).
`basis_asset`	Shared basis asset.
`field_operator`	Subject field operator satisfying the contract documented in `encode_operator()`.
`observation_operator`	Legacy alias for `field_operator`.
`mask`	Target-domain mask for volumetric reconstruction.
`domain`	Optional target domain for non-volumetric reconstruction.
`support`	Optional target support for non-volumetric reconstruction.
`coeff_analysis`	Optional analysis-coordinate coefficient matrix (time x k).
`analysis_transform`	Optional transform descriptor from raw to analysis coordinates.
`offset`	Optional voxel offset vector in target space.
`run_info`	Optional run metadata.
`meta`	Optional metadata list.

Details

'transport_latent()' stores the 'fmrilatent' side of the operator-backed workflow: shared basis asset, subject field operator, raw and analysis-space coefficients, and decoder views for native or template projection. It does not fit GLMs or carry statistical summaries. Downstream model-fitting code should ordinarily consume `coef_time(x, coordinates = "analysis")`, then return coefficient-space effects and covariance matrices for projection with `decode_coefficients()` and `decode_covariance()`.

Value

A `TransportLatent` object inheriting from `ImplicitLatent`.

---

validate_nested_parcellations

Validate that parcellation levels are nested

Description

Validate that parcellation levels are nested

Usage

```
validate_nested_parcellations(levels)
```

Arguments

`levels`	List of integer label vectors (same length) representing parcellations.

Value

Invisibly TRUE; stops with an error if nesting fails or lengths differ.

---

validate_neuroarchive_handoff_contract

Validate the fmrilatent-to-neuroarchive handoff contract

Description

Validate the fmrilatent-to-neuroarchive handoff contract

Usage

```
validate_neuroarchive_handoff_contract(x, error = TRUE)
```

Arguments

x	Object to validate.
error	If 'TRUE', throw on failure; otherwise return 'FALSE'.

Value

Invisibly, 'x' on success, or 'FALSE' when 'error = FALSE'.

---

validate_portable_linear_map

Validate an object against the portable linear-map contract

Description

Returns the normalized canonical form invisibly when x satisfies the portable_linear_map contract. The canonical return value lets callers use the result directly rather than paying normalization a second time via as_portable_linear_map(). Set error = FALSE to receive FALSE on failure instead of a stop.

Usage

```
validate_portable_linear_map(x, context = "portable linear map", error = TRUE)
```

Arguments

x	Object to validate.
context	Optional label used in error messages.
error	If TRUE (the default), signal an error on failure. If FALSE, return FALSE silently on failure.

Value

Invisibly: the normalized portable-linear-map list on success, or FALSE when error = FALSE and validation fails. On failure with error = TRUE (the default), an error is raised.

See Also

portable_linear_map, as_portable_linear_map.

Examples

```
op <- validate_portable_linear_map(matrix(1:4, 2, 2))
op$n_source
op$n_target
validate_portable_linear_map(list(forward = function(x) x), error = FALSE)
```

`validate_shared_component_contract`
Validate a shared component contract

Description

Validate a shared component contract

Usage

```
validate_shared_component_contract(x, error = TRUE)
```

Arguments

`x`	Object to validate.
`error`	If 'TRUE', throw on failure; otherwise return 'FALSE'.

Value

Invisibly, 'x' on success, or 'FALSE' when 'error = FALSE'.

`validate_template_protocol`
Validate the reusable template protocol

Description

Checks that a template asset exposes the method-neutral surface expected by shared-structure consumers: loadings, rank, support/domain, measure, roughness, decoder, and projection behavior.

Usage

```
validate_template_protocol(template, error = TRUE)
```

Arguments

`template`	Template object.
`error`	If 'TRUE', throw on failure; otherwise return 'FALSE'.

Value

Invisibly, a manifest describing the verified protocol.

`voxel_subset_to_gsp` *Convert voxel subset to an rgsp graph*

Description

Convert voxel subset to an rgsp graph

Usage

```
voxel_subset_to_gsp(mask, voxel_indices, k_neighbors = 6L)
```

Arguments

`mask`	LogicalNeuroVol or 3D logical array.
`voxel_indices`	Integer vector of voxel indices (mask order).
`k_neighbors`	Number of nearest neighbours for k-NN graph.

Value

'gsp_graph' object (from rgsp).

`wavelet_active_latent`
Active-pencil wavelet latent (CDF 5/3)

Description

Active-pencil wavelet latent (CDF 5/3)

Usage

```
wavelet_active_latent(
  X,
  mask,
  levels_space = 2L,
  levels_time = 0L,
  threshold = 0
)
```

Arguments

`X`	Numeric matrix (time x voxels in mask order) or 4D array.
`mask`	LogicalNeuroVol or 3D logical array.
`levels_space`	Integer spatial lifting levels (default 2).
`levels_time`	Integer temporal lifting levels (default 0 = none).
`threshold`	Optional hard threshold after space transform.

Value

An ‘ImplicitLatent‘ with decoder supporting ‘time_idx‘ and ‘roi_mask‘.

`wrap_decoded` *Wrap flat decoded outputs into a domain-native representation*

Description

Wrap flat decoded outputs into a domain-native representation

Usage

```
wrap_decoded(x, values, time_idx = NULL, space = c("native", "template"), ...)

## S4 method for signature 'ImplicitLatent'
wrap_decoded(x, values, time_idx = NULL, space = c("native", "template"), ...)

## S4 method for signature 'LatentNeuroSurfaceVector'
wrap_decoded(x, values, time_idx = NULL, space = c("native", "template"), ...)

## S4 method for signature 'BilatLatentNeuroSurfaceVector'
wrap_decoded(x, values, time_idx = NULL, space = c("native", "template"), ...)

## S4 method for signature 'BlockLatentNeuroVector'
wrap_decoded(x, values, time_idx = NULL, space = c("native", "template"), ...)

## S4 method for signature 'LatentNeuroVec'
wrap_decoded(x, values, time_idx = NULL, space = c("native", "template"), ...)
```

Arguments

`x`	A latent object.
`values`	Flat decoded values, typically as a vector or matrix.
`time_idx`	Optional integer time indices associated with `values`.
`space`	Output space to wrap into.
`...`	Additional arguments passed to methods.

Value

Domain-native wrapped representation.

Index