Package ‘neuroim2’

January 14, 2026

Type Package

Title Data Structures for Brain Imaging Data

Version 0.8.5

LinkingTo Rcpp, RcppArmadillo, RcppParallel

Author Bradley R Buchsbaum [aut, cre, cph]

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

Description A collection of data structures and methods for handling volumetric brain imaging data, with a focus on functional magnetic resonance imaging (fMRI). Provides efficient representations for three-dimensional and four-dimensional neuroimaging data through sparse and dense array implementations, memory-mapped file access for large datasets, and spatial transformation capabilities. Implements methods for image resampling, spatial filtering, region of interest analysis, and connected component labeling. General introduction to fMRI analysis can be found in Poldrack et al. (2024, ``Handbook of functional MRI data analysis'', <ISBN:9781108795760>).

License MIT + file LICENSE

Encoding UTF-8

VignetteBuilder knitr, rmarkdown

Depends R (>= 4.3.0), Matrix

Collate 'RcppExports.R' 'all_generic.R' 'all_class.R' 'SparseNeuroVec-validity.R' 'afni_io.R' 'array_like.R' 'axis.R' 'big_neurovec.R' 'binary_io.R' 'cgb.R' 'cgb_nuisance.R' 'clustered_neurovec.R' 'clustervol.R' 'common.R' 'conncomp.R' 'downsample.R' 'file_format.R' 'filebacked_neurovec.R' 'globals.R' 'index_vol.R' 'mapped_neurovec.R' 'meta_info.R' 'meta_info_api.R' 'neuro_obj.R' 'neurohypervec.R' 'neuroim.R' 'neuroslice.R' 'neurospace.R' 'neurovec.R' 'neurovecseq.R' 'neurovol.R' 'nifti_extensions.R' 'nifti_io.R' 'niml_io.R' 'ops.R' 'palettes.R' 'plot-helpers.R' 'plot-montage.R' 'plot-ortho.R' 'plot-overlay.R' 'read_image.R' 'resample.R' 'resample_to.R' 'roi.R' 'searchlight.R' 'simulate.R' 'sparse_neurovec.R' 'spat_filter.R' 'theme.R' 'zzz.R'

RoxygenNote 7.3.3

Imports purrr, assertthat, mmap, Rcpp, RcppParallel, RNifti, dbscan, stringr, methods, bigstatsr, RNiftyReg, future.apply, deflist, crayon, ggplot2, magrittr, grid

Suggests testthat, covr, knitr, roxygen2, rmarkdown, Gmedian, R.utils, spelling, xml2

URL `https://github.com/bbuchsbaum/neuroim2`, `https://bbuchsbaum.github.io/neuroim2/`

BugReports `https://github.com/bbuchsbaum/neuroim2/issues`

Language en-US

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-01-14 20:30:02 UTC

Contents

`neuroim2-package` *neuroim2: neuroimaging data structures for analysis*

Description

The neuroim2 package provides tools and functions for analyzing and manipulating neuroimaging data. It supports various neuroimaging formats and offers a range of analysis techniques.

Main functions

- `read_vol`: Read neuroimaging volumes
- `write_vol`: Write neuroimaging volumes
- `NeuroVol`: Create NeuroVol objects
- `NeuroVec`: Create NeuroVec objects

Author(s)

Maintainer: Bradley R Buchsbaum `<brad.buchsbaum@gmail.com>` [copyright holder]

See Also

Useful links:

- `https://github.com/bbuchsbaum/neuroim2`
- `https://bbuchsbaum.github.io/neuroim2/`
- Report bugs at `https://github.com/bbuchsbaum/neuroim2/issues`

`AbstractSparseNeuroVec-class` *AbstractSparseNeuroVec Class*

Description

An abstract base class for sparse four-dimensional brain image representations. This class provides the foundation for efficient storage and manipulation of large, sparse neuroimaging data.

Details

The AbstractSparseNeuroVec class serves as a template for implementing various sparse representations of 4D brain images. It combines the spatial properties of `NeuroVec` with the efficiency of sparse data structures.

Slots

- `mask` An object of class `LogicalNeuroVol` defining the sparse domain of the brain image. This mask indicates which voxels contain non-zero data.
- `map` An object of class `IndexLookupVol` used to map between spatial coordinates and index/row coordinates in the sparse representation.

Subclasses

Concrete implementations of this abstract class should provide specific data storage mechanisms and methods for efficient access and manipulation of sparse 4D brain image data.

See Also

`NeuroVec-class` for the parent class. `LogicalNeuroVol-class` for the mask representation. `IndexLookupVol-class` for the spatial-to-index mapping.

`add_dim` *Add a Dimension to an Object*

Description

This function adds a new dimension to a given object, such as a matrix or an array.

Usage

```
add_dim(x, n)

## S4 method for signature 'NeuroSpace,numeric'
add_dim(x, n)
```

Arguments

`x`	The NeuroSpace object
`n`	Numeric value specifying the size of the new dimension

Value

An object of the same class as `x` with the new dimension added.

Examples

```
# Create a NeuroSpace object
x <- NeuroSpace(c(10, 10, 10), c(1, 1, 1))

# Add a new dimension with size 10
x1 <- add_dim(x, 10)
```

```
# Check the new dimension
ndim(x1) == 4
dim(x1)[4] == 10
```

`anatomical_axes` *Pre-defined anatomical axes*

Description

These constants define standard anatomical axes used in neuroimaging. Each axis has a defined direction vector in 3D space.

Usage

```
LEFT_RIGHT

RIGHT_LEFT

ANT_POST

POST_ANT

INF_SUP

SUP_INF
```

Format

An object of class `NamedAxis` of length 1.

An object of class `NamedAxis` of length 1.

An object of class `NamedAxis` of length 1.

An object of class `NamedAxis` of length 1.

An object of class `NamedAxis` of length 1.

An object of class `NamedAxis` of length 1.

`annotate_orientation` *Add L/R and A/P/S/I annotations (optional)*

Description

Add L/R and A/P/S/I annotations (optional)

Usage

```
annotate_orientation(
  plane = c("axial", "coronal", "sagittal"),
  dims,
  gp = grid::gpar(col = "white", cex = 0.9, fontface = "bold")
)
```

Arguments

`plane`	"axial", "coronal", or "sagittal"
`dims`	c(nrow, ncol) of the slice matrix
`gp`	grid::gpar style

Value

A ggplot2 layer with annotation_custom grobs

`Arith,NeuroVec,NeuroVol-method` *Arithmetic Operations for NeuroVec and NeuroVol*

Description

This function performs arithmetic operations on a NeuroVec object and a NeuroVol object.

Usage

```
## S4 method for signature 'NeuroVec,NeuroVol'
Arith(e1, e2)
```

Arguments

`e1`	A NeuroVec object.
`e2`	A NeuroVol object.

Value

A DenseNeuroVec object resulting from the arithmetic operation.

Arith,NeuroVol,NeuroVec-method

Arithmetic Operations for NeuroVol and NeuroVec

Description

This function performs arithmetic operations on a NeuroVol object and a NeuroVec object.

Usage

```
## S4 method for signature 'NeuroVol,NeuroVec'
Arith(e1, e2)
```

Arguments

`e1`	A NeuroVol object.
`e2`	A NeuroVec object.

Value

A DenseNeuroVec object resulting from the arithmetic operation.

Arith,ROIVol,ROIVol-method

This function performs arithmetic operations on two ROIVol objects.

Description

This function performs arithmetic operations on two ROIVol objects.

Usage

```
## S4 method for signature 'ROIVol,ROIVol'
Arith(e1, e2)
```

Arguments

`e1`	An ROIVol object.
`e2`	An ROIVol object.

Value

An ROIVol object resulting from the arithmetic operation.

`Arith-methods` *Arithmetic Operations*

Description

Methods for performing arithmetic operations on neuroimaging objects

This method performs arithmetic operations between two ROIVol objects (`e1` and `e2`) using a generic arithmetic function. The dimensions of both objects are checked for compatibility before performing the operation.

Perform an arithmetic operation between two DenseNeuroVec objects. The input DenseNeuroVec objects must have the same dimensions and NeuroSpace objects. The method computes the elementwise arithmetic operation and returns a new DenseNeuroVec object.

Perform an arithmetic operation between a SparseNeuroVol object and a NeuroVol object. The input SparseNeuroVol and NeuroVol objects must have the same dimensions. The method performs the arithmetic operation on the non-zero values of the SparseNeuroVol and the corresponding values of the NeuroVol. The result is returned as a new DenseNeuroVol object.

Perform an arithmetic operation between a NeuroVol object and a SparseNeuroVol object. The input NeuroVol and SparseNeuroVol objects must have the same dimensions. The method performs the arithmetic operation on the values of the NeuroVol and the non-zero values of the SparseNeuroVol. The result is returned as a new DenseNeuroVol object.

Perform an arithmetic operation between two NeuroVec objects. The input NeuroVec objects must have the same dimensions. The method performs the arithmetic operation on the elements of the NeuroVec objects. The result is returned as a new DenseNeuroVec object.

Usage

```
## S4 method for signature 'SparseNeuroVol,SparseNeuroVol'
Arith(e1, e2)

## S4 method for signature 'DenseNeuroVol,DenseNeuroVol'
Arith(e1, e2)

## S4 method for signature 'DenseNeuroVec,DenseNeuroVec'
Arith(e1, e2)

## S4 method for signature 'SparseNeuroVol,NeuroVol'
Arith(e1, e2)

## S4 method for signature 'NeuroVol,SparseNeuroVol'
Arith(e1, e2)

## S4 method for signature 'SparseNeuroVec,SparseNeuroVec'
Arith(e1, e2)

## S4 method for signature 'NeuroVec,NeuroVec'
Arith(e1, e2)
```

Arguments

`e1`	A NeuroVec object.
`e2`	A NeuroVec object.

Value

A SparseNeuroVol object representing the result of the arithmetic operation.

An ROIVol object containing the result of the arithmetic operation between `e1` and `e2`.

A DenseNeuroVec object representing the result of the arithmetic operation.

A DenseNeuroVol object representing the result of the arithmetic operation.

A DenseNeuroVol object representing the result of the arithmetic operation.

A DenseNeuroVec object representing the result of the arithmetic operation.

`ArrayLike3D-class` *ArrayLike3D Class*

Description

A virtual class for representing three-dimensional array-like objects. It provides a common interface for 3D array operations.

`ArrayLike4D-class` *ArrayLike4D Class*

Description

A virtual class for representing four-dimensional array-like objects. It is intended to serve as a base class for 4D array representations.

`ArrayLike5D-class` *ArrayLike5D Class*

Description

A virtual class for representing five-dimensional array-like objects. This class serves as an interface for objects that mimic 5D arrays.

`as` *conversion from* `NeuroVol` *to* `LogicalNeuroVol`

Description

This function provides a method to coerce an object of class `ROIVec` to a `matrix`.

This function provides a method to coerce an object of class `ROIVol` to a `DenseNeuroVol`.

Arguments

`from`	An object of class `ROIVol` to be coerced to a `DenseNeuroVol`.

Value

A `matrix` obtained by coercing the `ROIVec` object.

A `DenseNeuroVol` object obtained by coercing the `ROIVol` object.

`as-ClusteredNeuroVol-DenseNeuroVol`

Convert ClusteredNeuroVol to DenseNeuroVol

Description

This method converts a ClusteredNeuroVol into an equivalent DenseNeuroVol object.

Arguments

`from`	A `ClusteredNeuroVol` object to be converted

Details

Convert a ClusteredNeuroVol Object to a DenseNeuroVol Object

Value

A `DenseNeuroVol` object

See Also

`ClusteredNeuroVol`, `DenseNeuroVol`

Examples

```
# Create a clustered volume
mask <- read_vol(system.file("extdata", "global_mask_v4.nii", package="neuroim2"))
clusters <- rep(1:5, length.out=sum(mask))
cvol <- ClusteredNeuroVol(mask, clusters)

# Convert to DenseNeuroVol
dvol <- as(cvol, "DenseNeuroVol")
```

`as.array` *Generic as.array Method*

Description

Coerces an object to a base array using S4 dispatch when available.

Usage

```
as.array(x, ...)
```

Arguments

`x`	An object to be coerced to an array.
`...`	Additional arguments passed to methods.

Value

An `array` representation of the input `x`.

`as.array,ClusteredNeuroVol-method` *Convert ClusteredNeuroVol to a base array*

Description

Ensures that clustered volumes dispatch through the ‘as.array‘ S4 generic and return dense arrays of cluster labels aligned to the underlying space.

Usage

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

Arguments

`x`	A ‘ClusteredNeuroVol‘ instance.
`...`	Additional arguments (currently ignored).

Value

A dense array of cluster ids.

`as.array,SparseNeuroVol-method` *Convert SparseNeuroVol to a base array*

Description

Provides an ‘as.array‘ S4 method so sparse volumes can be coerced with the same syntax used for dense objects.

Usage

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

Arguments

`x`	A ‘SparseNeuroVol‘ instance.
`...`	Additional arguments (currently ignored).

Value

A dense array with voxel values at their spatial locations and zeros elsewhere.

`as.dense` *Convert to dense representation*

Description

Convert to dense representation

Usage

```
as.dense(x)
```

Arguments

`x`	the object to densify

Value

A dense representation of the input object.

Examples

```
# Create a sparse representation
space <- NeuroSpace(c(10,10,10,4), c(1,1,1))
mask <- array(runif(10*10*10) > 0.8, c(10,10,10))  # ~20% of voxels active
data <- matrix(rnorm(sum(mask) * 4), 4, sum(mask))  # Random data for active voxels
sparse_vec <- SparseNeuroVec(data, space, mask)

# Convert to dense representation
dense_vec <- as.dense(sparse_vec)
# The dense representation has the same dimensions but stores all voxels
identical(dim(sparse_vec), dim(dense_vec))
```

`as.dense,ClusteredNeuroVol-method`

Coerce SparseNeuroVol to DenseNeuroVol

Description

Convert a sparse volumetric image to a dense representation with the same spatial geometry. Non-zero values stored in the sparse vector are placed at their corresponding linear indices in the dense array; all other voxels are 0.

This function provides a method to coerce an object of class `ROIVol` to a `DenseNeuroVol` using the `as.dense` method.

Usage

```
## S4 method for signature 'ClusteredNeuroVol'
as.dense(x)

## S4 method for signature 'SparseNeuroVol'
as.dense(x)

## S4 method for signature 'ROIVol'
as.dense(x)

## S4 method for signature 'SparseNeuroVec'
as.dense(x)
```

Arguments

`x`	An object of class `ROIVol` to be coerced to a `DenseNeuroVol`.

Value

A `NeuroVol` object representing the dense version of the clustered volume.

A DenseNeuroVol with identical spatial dimensions and values expanded from the sparse representation.

A `DenseNeuroVol` object obtained by coercing the `ROIVol` object.

`as.list,FileBackedNeuroVec-method`

Convert FileBackedNeuroVec to List

Description

Converts a FileBackedNeuroVec object to a list of DenseNeuroVol objects.

convert SparseNeuroVec to list of `DenseNeuroVol`

Usage

```
## S4 method for signature 'FileBackedNeuroVec'
as.list(x)

## S4 method for signature 'NeuroVec'
as.list(x)

## S4 method for signature 'SparseNeuroVec'
as.list(x)
```

Arguments

`x`	the object

Details

This method creates a deferred list, where each element is a DenseNeuroVol object representing a single volume from the FileBackedNeuroVec.

Value

A list of DenseNeuroVol objects

`as.logical,NeuroVol-method`
as.logical

Description

Convert NeuroVol to `LogicalNeuroVol`

Usage

```
## S4 method for signature 'NeuroVol'
as.logical(x)

## S4 method for signature 'ROIVol'
as.logical(x)
```

Arguments

`x`	the object

Details

the image values will be converted to using R base function `as.logical` and wrapped in `LogicalNeuroVol`

Value

an instance of `LogicalNeuroVol`

`as.mask` *Convert to a LogicalNeuroVol*

Description

Convert to a LogicalNeuroVol

Usage

```
as.mask(x, indices)
```

Arguments

`x`	the object to binarize
`indices`	the indices to set to TRUE

Value

A `LogicalNeuroVol` object with `TRUE` values at the specified `indices`.

Examples

```
# Create a simple 3D volume with random values
space <- NeuroSpace(c(10,10,10), spacing=c(1,1,1))
vol <- NeuroVol(array(runif(1000), c(10,10,10)), space)

# Create a mask by thresholding (values > 0.5 become TRUE)
mask1 <- as.mask(vol > 0.5)

# Create a mask by specifying indices
indices <- which(vol > 0.8)  # get indices of high values
mask2 <- as.mask(vol, indices)

# Both masks are LogicalNeuroVol objects
identical(class(mask1), class(mask2))
```

`as.mask,NeuroVol,missing-method` *Convert NeuroVol to a mask*

Description

This method converts a NeuroVol object to a mask by setting all positive values to TRUE and all non-positive values to FALSE.

This method converts a NeuroVol object to a mask by setting the specified indices to TRUE and the remaining elements to FALSE.

Usage

```
## S4 method for signature 'NeuroVol,missing'
as.mask(x)

## S4 method for signature 'NeuroVol,numeric'
as.mask(x, indices)
```

Arguments

`x`	A NeuroVol object to convert to a mask.
`indices`	A numeric vector containing the indices of the input NeuroVol that should be set to TRUE in the resulting mask.

Value

A LogicalNeuroVol object representing the mask created from the input NeuroVol.

A LogicalNeuroVol object representing the mask created from the input NeuroVol with specified indices.

`as.matrix` *Generic as.matrix Method*

Description

Coerces an object to a matrix.

Usage

```
as.matrix(x, ...)
```

Arguments

`x`	An object to be coerced to a matrix.
`...`	Additional arguments passed to methods.

Value

A `matrix` representation of the input `x`.

`as.matrix,ClusteredNeuroVec-method`
convert a `NeuroVec` *to a matrix*

Description

convert a `NeuroVec` to a matrix

convert a `ROIVec` to a matrix

Convert to Matrix

Usage

```
## S4 method for signature 'ClusteredNeuroVec'
as.matrix(x, by = c("cluster", "voxel"))

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

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

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

## S4 method for signature 'ROIVec'
```

```
as.matrix(x)

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

Arguments

`x`	The object to convert to a matrix
`by`	For ClusteredNeuroVec: controls the conversion target. Defaults to "cluster" to return a TxK matrix of cluster time-series. "voxel" is reserved for future use.
`...`	Additional arguments

Value

A matrix representation of the object

`as.numeric,SparseNeuroVol-method`

Convert SparseNeuroVol to numeric

Description

Convert SparseNeuroVol to numeric

Usage

```
## S4 method for signature 'SparseNeuroVol'
as.numeric(x)

## S4 method for signature 'ROIVol'
as.numeric(x)
```

Arguments

`x`	the object to convert

Value

A numeric vector of length `nrow(x@coords)`

`as.raster` *Generic Method for Converting Objects to Raster Format*

Description

Converts an object to a raster (bitmap) representation.

Arguments

`x`	An object to be converted.
`...`	Additional arguments passed to the conversion methods.

Value

A `raster` object representing `x`.

`as.sparse` *Convert to from dense to sparse representation*

Description

Convert to from dense to sparse representation

Usage

```
as.sparse(x, mask, ...)
```

Arguments

`x`	the object to make sparse, e.g. `DenseNeuroVol` or `DenseNeuroVec`
`mask`	the elements to retain
`...`	additional arguments

Details

`mask` can be an integer vector of 1D indices or a mask volume of class `LogicalNeuroVol`

Value

A sparse representation of the input object, containing only the elements specified by `mask`.

Examples

```
bvol <- NeuroVol(array(runif(24*24*24), c(24,24,24)), NeuroSpace(c(24,24,24), c(1,1,1)))
indmask <- sort(sample(1:(24*24*24), 100))
svol <- as.sparse(bvol, indmask)


mask <- LogicalNeuroVol(runif(length(indmask)), space=space(bvol), indices=indmask)
sum(mask) == 100
```

`as.sparse,DenseNeuroVec,LogicalNeuroVol-method`

Convert DenseNeuroVec to sparse representation using mask

Description

This method converts a DenseNeuroVec object to a sparse representation using a given LogicalNeuroVol mask.

This method converts a DenseNeuroVec object to a sparse representation using a given numeric mask.

Usage

```
## S4 method for signature 'DenseNeuroVec,LogicalNeuroVol'
as.sparse(x, mask)

## S4 method for signature 'DenseNeuroVec,numeric'
as.sparse(x, mask)

## S4 method for signature 'DenseNeuroVol,LogicalNeuroVol'
as.sparse(x, mask)

## S4 method for signature 'DenseNeuroVol,numeric'
as.sparse(x, mask)

## S4 method for signature 'ROIVol,ANY'
as.sparse(x)
```

Arguments

`x`	A DenseNeuroVec object to convert to a sparse representation.
`mask`	A numeric vector representing the mask to apply during conversion.

Value

A SparseNeuroVec object resulting from the conversion.

A SparseNeuroVec object resulting from the conversion.

`as.vector,SparseNeuroVol-method`
Convert SparseNeuroVol to a base vector

Description

Supplies an ‘as.vector‘ S4 method that flattens sparse volumes to a dense vector, keeping the same voxel ordering as ‘as.array‘.

Usage

```
## S4 method for signature 'SparseNeuroVol'
as.vector(x, mode = "any")
```

Arguments

`x`	A ‘SparseNeuroVol‘ instance.
`mode`	Optional coercion mode (see [base::as.vector]).

Value

A vector of length ‘prod(dim(x))‘.

`as_mmap` *Convert a NeuroVec to a memory-mapped representation*

Description

Generic for converting neuroimaging vectors to a memory-mapped `MappedNeuroVec` on disk (when possible).

Methods for the `as_mmap` generic, which convert various neuroimaging vector types to a `MappedNeuroVec` backed by an on-disk NIfTI file.

Usage

```
as_mmap(x, file = NULL, ...)

## S4 method for signature 'MappedNeuroVec'
as_mmap(x, file = NULL, ...)

## S4 method for signature 'FileBackedNeuroVec'
as_mmap(x, file = NULL, ...)

## S4 method for signature 'NeuroVec'
as_mmap(x, file = NULL, data_type = "FLOAT", overwrite = FALSE, ...)
```

```
## S4 method for signature 'SparseNeuroVec'
as_mmap(x, file = NULL, data_type = "FLOAT", overwrite = FALSE, ...)
```

Arguments

`x`	A neuroimaging vector (`NeuroVec`, `MappedNeuroVec`, or `FileBackedNeuroVec`).
`file`	Optional output file name. If `NULL`, a temporary file with extension `.nii` is created.
`...`	Additional arguments passed to methods (e.g. `data_type`, `overwrite`).
`data_type`	Character string specifying the output data type for the NIfTI file. Should be one of: "BINARY", "UBYTE", "SHORT", "INT", "FLOAT", "DOUBLE". Default is "FLOAT".
`overwrite`	Logical; if `TRUE`, overwrite an existing file at the specified path. Default is `FALSE`.

Value

A `MappedNeuroVec` (or `x` itself if already memory-mapped).

A `MappedNeuroVec` (or `x` itself if it is already memory-mapped).

`as_nifti_header` *Construct a Minimal NIfTI-1 Header from a NeuroVol*

Description

Given a `NeuroVol` object (or similar), this function builds a basic NIfTI-1 header structure, populating essential fields such as `dim`, `pixdim`, `datatype`, the affine transform, and the quaternion parameters.

Usage

```
as_nifti_header(
  vol,
  file_name,
  oneFile = TRUE,
  data_type = "FLOAT",
  extensions = NULL
)
```

Arguments

`vol`	A `NeuroVol` (or 3D array-like) specifying dimensions, spacing, and affine transform.
`file_name`	A character string for the file name (used within the header but not necessarily to write data).

`oneFile`	Logical; if `TRUE`, sets the NIfTI magic to `"n+1"`, implying a single-file format (`.nii`). If `FALSE`, uses `"ni1"` (header+image).
`data_type`	Character specifying the data representation, e.g. `"FLOAT"`, `"DOUBLE"`. The internal code picks an integer NIfTI code.
`extensions`	Optional `NiftiExtensionList-class` object or list of `NiftiExtension-class` objects to include in the header.

Details

This is a convenience function that calls `createNIfTIHeader` first, then updates the fields (dimensions, `pixdim`, orientation, etc.) based on the `vol` argument. The voxel offset is set to 352 bytes (or larger if extensions are provided), and the quaternion is derived from the transform matrix via `matrixToQuatern`.

Note: This function primarily sets up a minimal header suitable for writing standard single-file NIfTI-1. If you need a more comprehensive or advanced usage, consider manually editing the returned list.

Value

A `list` representing the NIfTI-1 header fields, containing elements like `dimensions`, `pixdim`, `datatype`, `qform`, `quaternion`, `qfac`, `extensions`, etc. This can be passed to other functions that write or manipulate the header.

See Also

`createNIfTIHeader` for the base constructor of an empty NIfTI header. `NiftiExtension` for creating extensions.

`axes` *Extract Image Axes*

Description

Extract Image Axes

Usage

```
axes(x)

## S4 method for signature 'NeuroSpace'
axes(x)
```

Arguments

`x`	an object with a set of axes

Value

An object representing the `axes` of `x`.

Examples

```
x <- NeuroSpace(c(10,10,10), spacing=c(1,1,1))
class(axes(x)) == "AxisSet3D"
```

`AxisSet-class` *AxisSet*

Description

Virtual base class representing an ordered set of named axes.

Slots

`ndim` the number of axes (or dimensions)

`AxisSet1D-class` *AxisSet1D*

Description

A one-dimensional axis set

Slots

`i` the first axis

`AxisSet2D-class` *AxisSet2D*

Description

A two-dimensional axis set representing an ordered pair of named axes.

Slots

- `i` The first axis, inherited from AxisSet1D
- `j` The second axis, of class "NamedAxis"

See Also

`AxisSet1D-class`, `AxisSet3D-class`

Examples

```
# Create an AxisSet2D object
axis1 <- new("NamedAxis", axis = "x", direction = 1)
axis2 <- new("NamedAxis", axis = "y", direction = 1)
axisSet2D <- new("AxisSet2D", i = axis1, j = axis2, ndim = 2L)
```

`AxisSet3D-class` *AxisSet3D Class*

Description

A class representing a three-dimensional axis set, extending the AxisSet2D class with an additional third axis.

Slots

- `k` A `NamedAxis` object representing the third axis.

See Also

`AxisSet2D-class`, `NamedAxis-class`

Examples

```
# Create NamedAxis objects for each dimension
x_axis <- new("NamedAxis", axis = "x", direction = 1)
y_axis <- new("NamedAxis", axis = "y", direction = 1)
z_axis <- new("NamedAxis", axis = "z", direction = 1)

# Create an AxisSet3D object
axis_set_3d <- new("AxisSet3D", i = x_axis, j = y_axis, k = z_axis, ndim = 3L)
```

`AxisSet4D-class` *AxisSet4D Class*

Description

A class representing a four-dimensional axis set, extending the AxisSet3D class with an additional fourth axis.

Slots

`l` A `NamedAxis` object representing the fourth axis.

See Also

`AxisSet3D-class`, `NamedAxis-class`

Examples

```
# Create NamedAxis objects for each dimension
x_axis <- new("NamedAxis", axis = "x", direction = 1)
y_axis <- new("NamedAxis", axis = "y", direction = 1)
z_axis <- new("NamedAxis", axis = "z", direction = 1)
t_axis <- new("NamedAxis", axis = "t", direction = 1)

# Create an AxisSet4D object
axis_set_4d <- new("AxisSet4D", i = x_axis, j = y_axis, k = z_axis,
                   l = t_axis, ndim = 4L)
```

`AxisSet5D-class` *AxisSet5D Class*

Description

A class representing a five-dimensional axis set, extending the AxisSet4D class with an additional fifth axis.

Slots

`m` A `NamedAxis` object representing the fifth axis.

See Also

`AxisSet4D-class`, `NamedAxis-class`

Examples

```
# Create NamedAxis objects for each dimension
x_axis <- new("NamedAxis", axis = "x", direction = 1)
y_axis <- new("NamedAxis", axis = "y", direction = 1)
z_axis <- new("NamedAxis", axis = "z", direction = 1)
t_axis <- new("NamedAxis", axis = "t", direction = 1)
v_axis <- new("NamedAxis", axis = "v", direction = 1)

# Create an AxisSet5D object
axis_set_5d <- new("AxisSet5D", i = x_axis, j = y_axis, k = z_axis,
                   l = t_axis, m = v_axis, ndim = 5L)
```

`BigNeuroVec` *Create a Memory-Mapped Neuroimaging Vector*

Description

Creates a BigNeuroVec object, which represents a large neuroimaging vector using memory-mapped file storage. This allows working with neuroimaging data that is too large to fit in memory.

Usage

```
BigNeuroVec(
  data,
  space,
  mask,
  label = "",
  type = c("double", "float", "integer"),
  backingfile = tempfile()
)
```

Arguments

`data`	The input data to be stored
`space`	A NeuroSpace object defining the spatial properties
`mask`	A logical mask indicating which voxels contain data
`label`	Optional character string label for the vector
`type`	Storage type, one of "double", "float", or "integer"
`backingfile`	Path to the file used for memory mapping (defaults to tempfile())

Value

A new BigNeuroVec object that provides memory-efficient access to large neuroimaging data through memory mapping. The object contains the spatial properties, mask, and memory-mapped data storage.

Examples

```
# Load an example 4D brain image
example_file <- system.file("extdata", "global_mask_v4.nii", package = "neuroim2")
example_4d_image <- read_vec(example_file)

# Create a mask (e.g., selecting voxels with values > 0)
mask <- array(as.vector(example_4d_image[,,,1]) > 0,
             dim = dim(example_4d_image)[1:3])

if(requireNamespace("bigstatsr", quietly = TRUE)) {
  # Create a BigNeuroVec with memory mapping
  big_vec <- BigNeuroVec(data = example_4d_image@.Data,
                        space = space(example_4d_image),
                        mask = mask,
                        label = "Example BigNeuroVec")
  print(big_vec)
}
```

`BigNeuroVec-class` *BigNeuroVec Class*

Description

A class representing a sparse four-dimensional brain image backed by a disk-based big matrix. BigNeuroVec objects are designed for efficient handling of large-scale brain imaging data that exceeds available memory.

Details

BigNeuroVec leverages file-backed storage to manage large 4D neuroimaging datasets that would typically exceed available RAM. It combines the sparse representation framework of `AbstractSparseNeuroVec` with the disk-based storage capabilities of `FBM`, allowing for out-of-core computations on massive datasets.

Slots

`data` An instance of class `FBM` from the `bigstatsr` package, containing time-series data. The FBM (File-Backed Big Matrix) is a matrix-like structure stored on disk, enabling efficient handling of large-scale data.

Inheritance

`BigNeuroVec` inherits from:

- `NeuroVec`: Base class for 4D brain images
- `AbstractSparseNeuroVec`: Provides sparse representation framework
- `ArrayLike4D`: Interface for 4D array-like operations

See Also

`AbstractSparseNeuroVec-class` for the parent sparse representation class. `NeuroVec-class` for the base 4D brain image class. `FBM` for details on File-Backed Big Matrix objects.

`bilateral_filter` *Apply a bilateral filter to a volumetric image*

Description

This function smooths a volumetric image (3D brain MRI data) using a bilateral filter. The bilateral filter considers both spatial closeness and intensity similarity for smoothing.

Usage

```
bilateral_filter(vol, mask, spatial_sigma = 2, intensity_sigma = 1, window = 1)
```

Arguments

`vol`	A `NeuroVol` object representing the image volume to be smoothed.
`mask`	An optional `LogicalNeuroVol` object representing the image mask that defines the region where the filtering is applied. If not provided, the entire volume is considered.
`spatial_sigma`	A numeric value specifying the standard deviation of the spatial Gaussian kernel (default is 2).
`intensity_sigma`	A numeric value specifying the standard deviation of the intensity Gaussian kernel (default is 25).
`window`	An integer specifying the number of voxels around the center voxel to include on each side. For example, window=1 for a 3x3x3 kernel (default is 1).

Value

A smoothed image of class `NeuroVol`.

Examples

```
brain_mask <- read_vol(system.file("extdata", "global_mask_v4.nii", package="neuroim2"))

# Apply bilateral filtering to the brain volume
filtered_vol <- bilateral_filter(brain_mask, brain_mask, spatial_sigma = 2,
intensity_sigma = 25, window = 1)
```

`bilateral_filter_4d` *Apply a 4D bilateral filter to a NeuroVec*

Description

This function applies a full 4D bilateral filter to a `NeuroVec`, smoothing jointly across space (x, y, z) and time (t). The filter uses spatial, temporal, and intensity kernels to preserve edges while reducing noise, leveraging a parallel C++ backend for performance.

Usage

```
bilateral_filter_4d(
  vec,
  mask,
  spatial_sigma = 2,
  intensity_sigma = 1,
  temporal_sigma = 1,
  spatial_window = 1,
  temporal_window = 1,
  temporal_spacing = 1
)
```

Arguments

`vec`	A `NeuroVec` object (4D image).
`mask`	An optional `LogicalNeuroVol` or `NeuroVol` specifying the spatial region to process. If omitted, the entire spatial extent is processed.
`spatial_sigma`	Numeric; standard deviation of the spatial Gaussian (default 2).
`intensity_sigma`	Numeric; standard deviation of the intensity Gaussian (default 1).
`temporal_sigma`	Numeric; standard deviation of the temporal Gaussian (default 1).
`spatial_window`	Integer; half-width of the spatial window in voxels (default 1), e.g., 1 => 3x3x3 spatial neighborhood.
`temporal_window`	Integer; half-width of the temporal window in frames (default 1), e.g., 1 => 3 timepoints (t-1, t, t+1).
`temporal_spacing`	Numeric; spacing of the temporal dimension (e.g., TR in seconds). Default is 1. This sets the temporal scale used for the temporal kernel.

Details

Parameter guidance and units: - spatial_sigma: Measured in physical units (millimeters). Distances are computed using `spacing(vec)[1:3]`, so choose `spatial_sigma` relative to voxel size. As a rule of thumb, set it to about 1-2 voxel sizes (e.g., 2-4 mm for 2 mm isotropic data) for moderate smoothing. - intensity_sigma: Dimensionless multiplier of the global intensity standard deviation. Internally, the filter uses exp(-(dI)^2 / (2 * (intensity_sigma * sigma_I)^2)), where sigma_I is the standard deviation of all finite voxel intensities within the mask across time. Start with 1.0 for moderate smoothing; use 0.5-0.8 to preserve more edges, or 1.5-2.0 for stronger smoothing. - temporal_sigma: Measured in `temporal_spacing` units (e.g., seconds). Typical values are 0.5-2 x TR. Larger values blend more across time.

Choosing the neighborhood window sizes: - spatial_window controls the discrete spatial support. A common choice is `ceiling(2 * spatial_sigma / min(spacing(vec)[1:3]))`, which covers ~95 - temporal_window similarly can be set to `ceiling(2 * temporal_sigma / temporal_spacing)`.

Quick presets (typical fMRI with 2-3 mm voxels and TR~2s): - Light: spatial_sigma = 1 x min(spacing), intensity_sigma = 0.8, temporal_sigma = 0.5 x TR, windows = 1 - Moderate (default-ish): spatial_sigma = 1.5 x min(spacing), intensity_sigma = 1.0, temporal_sigma = 1 x TR, windows = 1-2 - Strong: spatial_sigma = 2 x min(spacing), intensity_sigma = 1.5, temporal_sigma = 1.5 x TR, windows = 2

Tip: If your time axis has known TR, pass it via `temporal_spacing`. For NIfTI inputs, you can get TR via:

```
hdr <- read_header(nifti_path)
tr  <- hdr@header$pixdim[5]
out <- bilateral_filter_4d(vec, mask, temporal_spacing = tr)
```

Value

A `NeuroVec` with filtered data.

See Also

`bilateral_filter`, `NeuroVec-class`, `NeuroVol-class`

Examples

```
vec <- read_vec(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))
mask <- read_vol(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))
out  <- bilateral_filter_4d(vec, mask,
                            spatial_sigma = 2, intensity_sigma = 1,
                            temporal_sigma = 1, spatial_window = 1,
                            temporal_window = 1, temporal_spacing = 1)
```

`BinaryReader` *Create Binary Reader Object*

Description

Create a new instance of the BinaryReader class for reading bulk binary data.

Usage

```
BinaryReader(
  input,
  byte_offset,
  data_type,
  bytes_per_element,
  endian = .Platform$endian,
  signed = TRUE
)
```

Arguments

`input`	Character string (file name) or connection object to read from
`byte_offset`	Integer specifying bytes to skip at start of input
`data_type`	Character string specifying R data type ('integer', 'double', etc.)
`bytes_per_element`	Integer specifying bytes per data element (e.g., 4 or 8)
`endian`	Character string specifying endianness ('big' or 'little', default: platform-specific)
`signed`	Logical indicating if data type is signed (default: TRUE)

Value

An object of class BinaryReader

See Also

`BinaryWriter` for writing binary data

Examples

```
# Create a temporary binary file
tmp <- tempfile()
writeBin(rnorm(100), tmp, size = 8)


# Read from existing connection with offset
con <- file(tmp, "rb")
reader <- BinaryReader(con, byte_offset=0,
                       data_type = "DOUBLE", bytes_per_element = 8L)
close(reader)

# Clean up
unlink(tmp)
```

`BinaryReader-class` *BinaryReader Class*

Description

Class supporting reading of bulk binary data from a connection

Slots

`input` The binary input connection

`byte_offset` The number of bytes to skip at the start of input

`data_type` The data type of the binary elements

`bytes_per_element` The number of bytes in each data element (e.g. 4 or 8 for floating point numbers)

`endian` The endianness of the binary input connection

`signed` Logical indicating whether the data type is signed

`BinaryWriter` *Create Binary Writer Object*

Description

Create a new instance of the BinaryWriter class for writing bulk binary data.

Usage

```
BinaryWriter(
  output,
  byte_offset,
  data_type,
  bytes_per_element,
  endian = .Platform$endian
)
```

Arguments

`output`	Character string (file name) or connection object to write to
`byte_offset`	Integer specifying bytes to skip at start of output
`data_type`	Character string specifying R data type ('integer', 'double', etc.)
`bytes_per_element`	Integer specifying bytes per data element (e.g., 4 or 8)
`endian`	Character string specifying endianness ('big' or 'little', default: platform-specific)

Value

An object of class BinaryWriter

See Also

`BinaryReader` for reading binary data

Examples

```
tmp <- tempfile()
# Write to existing connection with offset
con <- file(tmp, "wb")
writer <- BinaryWriter(con, byte_offset = 100L,
                      data_type = "integer", bytes_per_element = 4L)
unlink(tmp)
```

`BinaryWriter-class` *BinaryWriter Class*

Description

This class supports writing of bulk binary data to a connection

Slots

`output` The binary output connection

`byte_offset` The number of bytes to skip at the start of input

`data_type` The data type of the binary elements

`bytes_per_element` The number of bytes in each data element (e.g. 4 or 8 for floating point numbers)

`endian` The endianness of the binary output connection

`bounds` *Extract Spatial Bounds of an Image*

Description

This function extracts the spatial bounds (origin + dim * spacing) of an image represented by the input object.

Usage

```
bounds(x)

## S4 method for signature 'NeuroSpace'
bounds(x)
```

Arguments

`x`	The object with the ‘bounds‘ property, typically an image.

Value

A numeric `matrix` with two columns specifying the min (column 1) and max (column 2) bounds of each dimension of `x`.

Examples

```
bspace <- NeuroSpace(c(10, 10, 10), c(2, 2, 2))
b <- bounds(bspace)
nrow(b) == ndim(bspace)
ncol(b) == 2
```

`centroid` *return the centroid of an object*

Description

return the centroid of an object

Usage

```
centroid(x, ...)

## S4 method for signature 'NeuroSpace'
centroid(x)

## S4 method for signature 'ROICoords'
centroid(x)
```

Arguments

`x`	an object with a centroid
`...`	extra args

Value

A numeric vector giving the centroid of `x`.

Examples

```
bspace <- NeuroSpace(c(10,10,10), c(2,2,2))
centroid(bspace)
```

`centroids` *Return a matrix of centroids of an object*

Description

Return a matrix of centroids of an object

Usage

```
centroids(x, ...)

## S4 method for signature 'ClusteredNeuroVec'
centroids(x, type = c("center_of_mass", "medoid"))

## S4 method for signature 'ClusteredNeuroVol'
centroids(x, type = c("center_of_mass", "medoid"))
```

Arguments

`x`	an object with multiple centroids (e.g. a `ClusteredNeuroVol`)
`...`	extra args
`type`	the type of center of mass: one of "center_of_mass" or "medoid"

Details

For ‘type = "center_of_mass"‘, returns arithmetic mean coordinates; for ‘"medoid"‘, returns the most central point.

Value

A numeric `matrix` where each row represents the coordinates of a centroid.

A matrix of coordinates where each row represents the centroid of a cluster.

`cgb_filter` *Correlation-guided bilateral filtering (convenience wrapper)*

Description

High-level interface that builds a correlation-guided bilateral (CGB) graph with sensible defaults (similar to the bilateral filter interface) and immediately applies it to smooth the data.

Usage

```
cgb_filter(
  runs,
  mask = NULL,
  spatial_sigma = 2,
  window = NULL,
  corr_map = c("power", "exp", "soft"),
  corr_param = 2,
  topk = 16L,
  passes = 1L,
  lambda = 1,
  leave_one_out = FALSE,
  run_weights = NULL,
  add_self = TRUE,
  time_weights = NULL,
  confounds = NULL,
  robust = c("none", "huber", "tukey"),
  robust_c = 1.345,
  return_graph = FALSE
)
```

Arguments

`runs`	A `NeuroVec` or a list of `NeuroVec`.
`mask`	Optional `LogicalNeuroVol`/`NeuroVol` or logical array for spatial masking. Defaults to in-mask voxels.
`spatial_sigma`	Spatial Gaussian sigma in mm. Used both for weighting and, when `window` is `NULL`, to auto-choose the neighborhood size.
`window`	Integer half-width of the cubic neighborhood. If `NULL`, it is computed as `ceiling(2 * spatial_sigma / min(spacing))` and at least 1.
`corr_map`	Mapping from pooled correlation to affinity; one of `"power"`, `"exp"`, or `"soft"`. Defaults to `"power"`.
`corr_param`	Parameter for `corr_map` (gamma/tau/r0 respectively).
`topk`	Keep strongest `k` neighbors (0 keeps all). Defaults to 16.
`passes`	Number of smoothing passes (>=1). Defaults to 1.
`lambda`	Blend factor in [0,1] per pass. Defaults to 1 (pure diffusion).

`leave_one_out`	If `TRUE` and multiple runs are supplied, builds LORO graphs and returns a list of smoothed runs.
`run_weights`	Optional numeric weights per run for Fisher-z pooling.
`add_self`	Logical; add a tiny self-edge before normalization.
`time_weights`	Optional list (or single vector) of per-run time weights.
`confounds`	Optional list (or single matrix) of per-run confounds.
`robust`	One of `"none"`, `"huber"`, or `"tukey"`.
`robust_c`	Tuning constant for robust weights.
`return_graph`	Logical; if `TRUE`, also return the graph(s) alongside the smoothed data.

Details

This is a convenience front-end to `cgb_make_graph` and `cgb_smooth` with a bilateral-like interface: - If `window` is `NULL`, it is chosen as `ceiling(2 * spatial_sigma / min(spacing))` (at least 1). Larger windows allow correlations over more distant neighbors, at the cost of extra compute and memory. - `spatial_sigma` (in mm) controls how quickly spatial weights fall with distance. Small values emphasize very local structure; larger values mix information over a wider spatial footprint. - `corr_map` and `corr_param` set how pooled correlations are turned into edge weights: * `"power"`: `a(r) = r^gamma` for `r > 0`. Larger `gamma` (e.g., 3-4) strongly emphasizes high correlations and produces more edge-preserving, patchy smoothing; smaller values (e.g., 1-2) behave more like standard correlation-weighted smoothing. * `"exp"`: Gaussian on `1 - r` with scale `tau`. Small `tau` keeps only very similar time-series; larger `tau` makes the filter closer to a spatial Gaussian while still respecting sign. * `"soft"`: `a(r) = max(r - r0, 0)`. Increasing `r0` discards more weak correlations and tends to sharpen edges but can make the result more piecewise-constant. - `topk` limits each voxel to at most `k` strongest neighbors. Smaller `topk` yields sparser, more anisotropic graphs (cheaper but sometimes less smooth); larger `topk` increases mixing and memory. - `passes` and `lambda` control diffusion strength. With `lambda = 1`, each pass applies pure graph diffusion; multiple passes compound smoothing. Choosing `lambda < 1` blends each pass with the identity and can prevent over-smoothing when using more passes. - Setting `leave_one_out = TRUE` for multi-run inputs builds a separate graph for each run that excludes its own correlations, which reduces information leakage in cross-validation or decoding workflows. - `time_weights`, `confounds`, and `robust`/`robust_c` adjust how time-points contribute to the correlation estimates. Down- weighting high-motion/high-DVARS frames (via `make_time_weights` and `robust != "none"`) will typically yield smoother, less noisy graphs but can also reduce effective temporal degrees of freedom. - Use `return_graph = TRUE` when you plan to reuse the constructed graph(s) with `cgb_smooth` or inspect their sparsity pattern.

Value

If `leave_one_out=FALSE`, a smoothed `NeuroVec`. If `leave_one_out=TRUE`, a list of smoothed `NeuroVec`. When `return_graph=TRUE`, returns a list with elements `result` and `graph` (single object or lists accordingly).

Examples

```
vec <- read_vec(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))
mask <- read_vol(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))
```

```
# Auto window from spatial_sigma and spacing, single pass
out <- cgb_filter(vec, mask, spatial_sigma = 3, window = NULL, topk = 16)

# Stronger diffusion with two passes and lambda < 1
out2 <- cgb_filter(vec, mask, spatial_sigma = 4, window = NULL,
                   passes = 2, lambda = 0.7)
```

`cgb_make_graph` *Build a correlation-guided bilateral (CGB) graph*

Description

Computes a sparse row-stochastic graph whose weights combine spatial proximity and pooled local time-series correlations. Supports optional censoring weights, nuisance regression via weighted QR projectors, leave-one-run-out graph construction, and robust down-weighting of high-DVARS volumes.

Usage

```
cgb_make_graph(
  runs,
  mask = NULL,
  window = 1L,
  spatial_sigma = 2,
  corr_map = c("power", "exp", "soft"),
  corr_param = 2,
  topk = 16L,
  leave_one_out = FALSE,
  run_weights = NULL,
  add_self = TRUE,
  time_weights = NULL,
  confounds = NULL,
  robust = c("none", "huber", "tukey"),
  robust_c = 1.345
)
```

Arguments

`runs`	A `NeuroVec` or a list of `NeuroVec` objects (typically one per run).
`mask`	Optional `LogicalNeuroVol`/`NeuroVol` or logical array defining in-mask voxels. Defaults to all in-mask voxels.
`window`	Integer half-width of the cubic spatial neighborhood (e.g., 1 yields a 3x3x3 window).
`spatial_sigma`	Spatial Gaussian sigma in mm.

`corr_map`	Mapping from pooled correlation to affinity; one of `"power"`, `"exp"`, or `"soft"`. The `"power"` and `"soft"` mappings rectify negative correlations, whereas `"exp"` preserves them (useful for sharpening more than smoothing).
`corr_param`	Parameter for the chosen `corr_map` (gamma, tau, or r0 respectively).
`topk`	Keep the strongest `k` neighbors after masking (0 keeps all).
`leave_one_out`	Logical; if `TRUE` and multiple runs are provided, returns a list of graphs where run `u` excludes its own correlations.
`run_weights`	Optional numeric weights per run used in Fisher-z pooling. Defaults to $n_k - 3$ (usable frames minus three) when omitted.
`add_self`	Logical; always inject a tiny self-edge before normalization.
`time_weights`	Optional list (or single vector) of per-run time weights $w_t \in [0, 1]$ applied before correlation estimation. An intercept is always included so correlations are computed on weighted, demeaned series.
`confounds`	Optional list (or single matrix) of per-run confound regressors to project out prior to correlation estimation.
`robust`	One of `"none"`, `"huber"`, or `"tukey"`; when not `"none"` an additional DVARS-style reweighting is applied.
`robust_c`	Tuning constant for the robust weights (Huber/Tukey).

Details

Graph construction overview: - Neighborhood: For each in-mask voxel i, consider a cubic spatial window of half-width `window` (i.e., (2*window+1)^3 candidates). Candidates outside the mask or bounds are ignored. - Spatial kernel: For a candidate j at physical distance d_ij (mm), assign a spatial weight w_s = exp(-d_ij^2 / (2 * spatial_sigma^2)). Distances use `spacing(spatial_space)`. - Correlation pooling: Compute Pearson correlation r_k(i,j) within each run k (optionally after nuisance projection/weights), transform to Fisher-z, pool across runs with weights ω_k (default $n_k - 3$), then back-transform to r_pool via tanh. - Correlation-to-affinity mapping (`corr_map`): * `"power"` (mode=0): a(r) = r^gamma for r>0 else 0. Parameter = gamma. * `"exp"` (mode=1): a(r) = exp(-(1 - r)^2 / (2 * tau^2)). Parameter = tau. * `"soft"` (mode=2): a(r) = max(r - r0, 0). Parameter = r0. - Combined weight: w_ij = w_s(i,j) * a(r_pool(i,j)). If `topk > 0`, keep the strongest `topk` neighbors. Optionally inject a small self-edge when `add_self=TRUE`. Finally, row-normalize to obtain a stochastic W.

Parameter guidance: - `spatial_sigma` is in mm. A typical choice is 1-2x the voxel size (e.g., 2-4 mm for 2 mm isotropic). Larger values increase spatial mixing. - `window` controls support; a good rule is `ceiling(2 * spatial_sigma / min(spacing))`. - `corr_map`: use `"power"` with `corr_param = 2` for robust smoothing; `"exp"` with `tau ~ 0.5-1.5` retains sign information; `"soft"` with `r0 ~ 0.1-0.3` thresholds weak correlations. - `topk`: 8-32 is a practical range; higher values densify the graph and increase compute/memory. - `leave_one_out`: for multi-run inputs, enabling this prevents a run from using its own correlations when building its graph (mitigates leakage).

Nuisance/time weights/robust options: - If `confounds`/`time_weights` specified (or `robust != "none"`), per-run weighted QR projectors are used; correlations are computed on the projected, weighted series. Robust options add DVARS-like down-weighting. - If the only request is an implicit intercept (no actual confounds, no time weights, no robust), the baseline builder is used for identical results.

Output and usage: - Returns CSR arrays (`row_ptr`, `col_ind`, `val`) that define a row-stochastic matrix W over masked voxels. Apply with `cgb_smooth`. - Complexity scales with number of masked voxels times neighborhood size (limited by `topk`). Memory proportional to number of retained edges.

Value

A list containing `row_ptr`, `col_ind`, `val`, `dims3d`, and `mask_idx`, or (if `leave_one_out=TRUE`) a list of such graphs.

Examples

```
vec <- read_vec(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))
mask <- read_vol(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))
# Build a graph with spatial sigma in mm and keep 16 neighbors
G <- cgb_make_graph(vec, mask, spatial_sigma = 3, window = 2, topk = 16)
# Smooth with one pass (pure diffusion)
sm <- cgb_smooth(vec, G, passes = 1, lambda = 1)
```

`cgb_smooth` *Apply a precomputed CGB graph to volumetric data*

Description

Apply a precomputed CGB graph to volumetric data

Usage

```
cgb_smooth(x, graph, passes = 1L, lambda = 1)
```

Arguments

`x`	A `NeuroVec` (4D) or `NeuroVol` (3D).
`graph`	Graph list returned by `cgb_make_graph`.
`passes`	Number of smoothing passes (>=1). Each pass multiplies by `W`; if `lambda` < `1` a simple diffusion blend `(1 - lambda)I + lambda W` is applied per pass.
`lambda`	Blend factor in $[0, 1]$ controlling diffusion strength.

Value

Smoothed object of the same class as `x`.

`cgb_smooth_loro` *Leave-one-run-out smoothing helper*

Description

Leave-one-run-out smoothing helper

Usage

```
cgb_smooth_loro(runs, graphs, passes = 1L, lambda = 1)
```

Arguments

`runs`	List of `NeuroVec` objects (one per run).
`graphs`	List of graphs returned by `cgb_make_graph(..., leave_one_out=TRUE)`.
`passes, lambda`	See `cgb_smooth`.

Value

A list of smoothed `NeuroVec` objects, one per run.

`close,BinaryReader-method` *Close a BinaryReader or BinaryWriter*

Description

Closes the underlying connection associated with a BinaryReader or BinaryWriter object. This should be called when you're done with the reader/writer to free system resources.

Usage

```
## S4 method for signature 'BinaryReader'
close(con)

## S4 method for signature 'BinaryWriter'
close(con)
```

Arguments

`con`	The BinaryReader or BinaryWriter object to close.

Value

Invisibly returns `NULL`, called for its side effect of closing the connection.

Examples

```
# Create a temporary file and write some data
tmp <- tempfile()
writer <- BinaryWriter(tmp, byte_offset = 0L,
                      data_type = "DOUBLE", bytes_per_element = 8L)
write_elements(writer, rnorm(100))
close(writer)

# Read the data back
reader <- BinaryReader(tmp, byte_offset = 0L,
                      data_type = "DOUBLE", bytes_per_element = 8L)
data <- read_elements(reader, 100)
close(reader)

# Clean up
unlink(tmp)
```

`ClusteredNeuroVec` *ClusteredNeuroVec: Cluster-aware 4D neuroimaging data*

Description

‘ClusteredNeuroVec‘ creates a 4D array-like object where voxels are grouped into clusters, with one time-series per cluster. All voxels within a cluster share the same time-series, making it ideal for parcellated analyses (e.g., Schaefer-Yeo parcellations).

Usage

```
ClusteredNeuroVec(x, cvol, FUN = mean, weights = NULL, label = "")
```

Arguments

`x`	Either a ‘NeuroVec‘ object to be reduced by clusters, or a pre-computed numeric matrix of cluster time-series (T x K, where T=time points, K=clusters)
`cvol`	A ‘ClusteredNeuroVol‘ object defining the cluster assignments
`FUN`	Reduction function to aggregate voxels within clusters (default: mean). Common choices include `mean`, `median`, or custom functions.
`weights`	Optional numeric vector of per-voxel weights for weighted aggregation. Must have length equal to the number of non-zero voxels in the mask.
`label`	Optional character label for the object (default: "")

Details

This class implements array-like 4D access while storing data efficiently as a TxK matrix instead of the full voxel x time representation. Each cluster's time-series is computed by applying the aggregation function (`FUN`) to all voxels within that cluster.

The object supports standard NeuroVec operations:

- Indexing: `x[,,,t]` to extract 3D volumes at time t
- Series extraction: `series(x, i, j, k)` for time-series at voxel (i,j,k)
- Matrix conversion: `as.matrix(x)` to get the TxK cluster matrix

Single-voxel clusters are handled efficiently without aggregation overhead.

Value

A `ClusteredNeuroVec` object containing:

cvol The input ClusteredNeuroVol defining cluster structure

ts A TxK matrix of cluster time-series (T=timepoints, K=clusters)

cl_map Integer vector mapping linear voxel indices to cluster IDs

label Character label for the object

See Also

`ClusteredNeuroVol` for creating cluster assignments, `cluster_searchlight_series` for cluster-based searchlight analysis, `series` for extracting time-series

Examples

```
# Create synthetic 4D data (10x10x10 volume, 20 timepoints)
sp4 <- NeuroSpace(c(10,10,10,20), c(1,1,1))
arr <- array(rnorm(10*10*10*20), dim=c(10,10,10,20))
vec <- NeuroVec(arr, sp4)

# Create a mask covering the central region
sp3 <- NeuroSpace(c(10,10,10), c(1,1,1))
mask_arr <- array(FALSE, dim=c(10,10,10))
mask_arr[3:8, 3:8, 3:8] <- TRUE
mask <- LogicalNeuroVol(mask_arr, sp3)

# Assign voxels to 5 random clusters
n_voxels <- sum(mask_arr)
clusters <- sample(1:5, n_voxels, replace=TRUE)
cvol <- ClusteredNeuroVol(mask, clusters)

# Create clustered representation
cv <- ClusteredNeuroVec(vec, cvol)

# Access like a regular NeuroVec
vol_t1 <- cv[,,,1]  # 3D volume at time 1
ts <- series(cv, 5, 5, 5)  # time-series at voxel (5,5,5)
```

```
# Get cluster time-series matrix
cluster_matrix <- as.matrix(cv)  # T x K matrix
dim(cluster_matrix)  # 20 x 5
```

`ClusteredNeuroVec-class` *ClusteredNeuroVec Class*

Description

A class representing a 4D neuroimaging dataset where voxels are grouped into clusters. Each cluster has a single time-series that is shared by all voxels within that cluster.

Slots

`cvol` A `ClusteredNeuroVol` object defining cluster assignments

`ts` A numeric matrix of dimensions T x K (time points x clusters)

`cl_map` An integer vector mapping each voxel to its cluster ID (0 for outside mask)

`label` A character string label for the object

`ClusteredNeuroVol-class` *ClusteredNeuroVol Class*

Description

This class represents a three-dimensional brain image divided into N disjoint partitions or clusters. It extends the `SparseNeuroVol` class to provide efficient storage and manipulation of clustered neuroimaging data.

Construct a `ClusteredNeuroVol` instance

Usage

```
ClusteredNeuroVol(mask, clusters, label_map = NULL, label = "")
```

Arguments

`mask`	an instance of class `LogicalNeuroVol`
`clusters`	a vector of clusters ids with length equal to number of nonzero voxels in mask `mask`
`label_map`	an optional `list` that maps from cluster id to a cluster label, e.g. (1 -> "FFA", 2 -> "PPA")
`label`	an optional `character` string used to label of the volume

Details

The ClusteredNeuroVol class is designed for efficient representation and manipulation of brain images with distinct, non-overlapping regions or clusters. It combines the memory efficiency of sparse representations with additional structures for managing cluster information.

The use case of `ClusteredNeuroVol` is to store volumetric data that has been clustered into discrete sets of voxels, each of which has an associated id. For example, this class can be used to represent parcellated neuroimaging volumes.

Value

`ClusteredNeuroVol` instance

Slots

`mask` A `LogicalNeuroVol` object representing the logical mask indicating the spatial domain of the set of clusters.

`clusters` An integer vector representing the cluster number for each voxel in the mask.

`label_map` A named list where each element represents a cluster and its name.

`cluster_map` An `environment` object that maps from cluster id to the set of 1D spatial indices belonging to that cluster.

Methods

This class inherits methods from the `SparseNeuroVol` class. Additional methods specific to cluster operations may be available.

Usage

ClusteredNeuroVol objects are particularly useful for:

- Representing parcellated brain images
- Storing results of clustering algorithms applied to neuroimaging data
- Efficient manipulation and analysis of region-based neuroimaging data

See Also

`SparseNeuroVol-class` for the parent sparse volume class. `LogicalNeuroVol-class` for the mask representation.

Examples

```
# Create a simple clustered brain volume
dim <- c(10L, 10L, 10L)
mask_data <- array(rep(c(TRUE, FALSE), 500), dim)
mask <- new("LogicalNeuroVol", .Data = mask_data,
            space = NeuroSpace(dim = dim, origin = c(0,0,0), spacing = c(1,1,1)))

clusters <- as.integer(runif(sum(mask_data)) * 5)+1
```

```
label_map <- list("Cluster1" = 1, "Cluster2" = 2, "Cluster3" = 3,
                  "Cluster4" = 4, "Cluster5" = 5)

cluster_map <- list()
for (i in 1:5) {
  cluster_map[[as.character(i)]] <- which(clusters == i)
}

clustered_vol <- ClusteredNeuroVol(
                     mask = mask,
                     clusters = clusters,
                     label_map = label_map)



# Create a simple space and volume
space <- NeuroSpace(c(16, 16, 16), spacing = c(1, 1, 1))
vol_data <- array(rnorm(16^3), dim = c(16, 16, 16))
vol <- NeuroVol(vol_data, space)

# Create a binary mask (e.g., values > 0)
mask_data <- vol_data > 0
mask_vol <- LogicalNeuroVol(mask_data, space)

# Get coordinates of masked voxels
mask_idx <- which(mask_data)
coords <- index_to_coord(mask_vol, mask_idx)

# Cluster the coordinates into 10 groups
set.seed(123)  # for reproducibility
kmeans_result <- kmeans(coords, centers = 10)

# Create the clustered volume
clustered_vol <- ClusteredNeuroVol(mask_vol, kmeans_result$cluster)

# Print information about the clusters
print(clustered_vol)
```

`clustered_searchlight` *Create a clustered searchlight iterator*

Description

This function generates a searchlight iterator that iterates over successive spatial clusters in an image volume. It allows for the exploration of spatially clustered regions within the provided mask by using either a pre-defined clustered volume or performing k-means clustering to generate the clusters.

Usage

```
clustered_searchlight(mask, cvol = NULL, csize = NULL)
```

Arguments

`mask`	A `NeuroVol` object representing the brain mask.
`cvol`	An optional `ClusteredNeuroVol` instance representing pre-defined clusters within the mask. If provided, the 'csize' parameter is ignored.
`csize`	An optional integer specifying the number of clusters to be generated using k-means clustering (ignored if `cvol` is provided).

Value

A `deferred_list` object containing `ROIVol` objects, each representing a clustered region within the image volume.

Examples

```
# Load an example brain mask
mask <- read_vol(system.file("extdata", "global_mask_v4.nii", package="neuroim2"))

# Generate a clustered searchlight iterator with 5 clusters
clust_searchlight <- clustered_searchlight(mask, csize = 5)
```

`cluster_searchlight_series`

Cluster-centroid searchlight over cluster time-series

Description

Iterate over clusters by their centroids and, for each seed cluster, return the time-series of the K nearest clusters (or those within a radius). This enables searchlight analysis at the cluster level rather than individual voxels.

Usage

```
cluster_searchlight_series(x, cvol = NULL, k = 10L, radius = NULL, label = "")
```

Arguments

`x`	A ‘ClusteredNeuroVec‘ object or a ‘NeuroVec‘ plus ‘cvol‘
`cvol`	A ‘ClusteredNeuroVol‘ (required if ‘x‘ is a ‘NeuroVec‘)
`k`	Integer, number of nearest clusters including the seed (default: 10). Will be capped at the total number of clusters if specified value exceeds it.
`radius`	Numeric distance in mm. If given, use all clusters within this radius instead of k-nearest neighbors. Cannot be used together with `k`.
`label`	Optional character label for the returned windows

Details

The function creates a searchlight around each cluster's centroid, selecting either:

- The `k` nearest clusters (when `k` is specified)
- All clusters within a given radius (when `radius` is specified)

This is particularly useful for cluster-level connectivity analyses or when working with parcellated data where voxel-level searchlights would be redundant.

Value

A list of `ROIVec` objects, one per cluster, where each ROIVec contains:

values A TxN matrix where T is the number of timepoints and N is the number of neighboring clusters (including the seed itself)

coords The centroid coordinates of the neighboring clusters

The seed cluster's time-series is always the first column in each ROIVec.

See Also

`ClusteredNeuroVec` for creating clustered neuroimaging vectors, `searchlight` for voxel-level searchlight analysis, `ROIVec` for the structure of returned windows

Examples

```
# Create synthetic 4D data (8x8x8 volume, 10 timepoints)
sp4 <- NeuroSpace(c(8,8,8,10), c(1,1,1))
arr <- array(rnorm(8*8*8*10), dim=c(8,8,8,10))
vec <- NeuroVec(arr, sp4)

# Create a mask covering most of the volume
sp3 <- NeuroSpace(c(8,8,8), c(1,1,1))
mask_arr <- array(FALSE, dim=c(8,8,8))
mask_arr[2:7, 2:7, 2:7] <- TRUE
mask <- LogicalNeuroVol(mask_arr, sp3)

# Assign voxels to 10 clusters
n_voxels <- sum(mask_arr)
clusters <- sample(1:10, n_voxels, replace=TRUE)
cvol <- ClusteredNeuroVol(mask, clusters)

# Create clustered representation
cv <- ClusteredNeuroVec(vec, cvol)

# Get cluster searchlight with 3 nearest neighbors
windows <- cluster_searchlight_series(cv, k = 3)
length(windows)  # 10 windows (one per cluster)

# Check first window
roi1 <- windows[[1]]
dim(values(roi1))  # 10 x 3 (timepoints x neighbors)
```

```
# Use radius-based neighborhoods (5mm radius)
windows_radius <- cluster_searchlight_series(cv, radius = 5)
# Each window may have different number of neighbors
```

`ColumnReader` *Create Column Reader Object*

Description

Create a new instance of the ColumnReader class for reading column-oriented data.

Usage

```
ColumnReader(nrow, ncol, reader)
```

Arguments

`nrow`	Integer specifying number of rows in data
`ncol`	Integer specifying number of columns in data
`reader`	Function that takes column indices and returns matrix

Value

An object of class ColumnReader

Examples

```
reader_func <- function(cols) {
  matrix(rnorm(100 * length(cols)), 100, length(cols))
}
col_reader <- ColumnReader(nrow = 100L, ncol = 10L, reader = reader_func)
```

`ColumnReader-class` *ColumnReader*

Description

A class that supports reading of data from a matrix-like storage format, such as a file or a database, in a column-wise manner.

Slots

- `nrow` An `integer` representing the number of rows in the matrix-like storage.
- `ncol` An `integer` representing the number of columns in the matrix-like storage.
- `reader` A `function` that takes a set of column indices as input and returns a `matrix` containing the requested columns from the storage.

`Compare-methods` *Comparison Operations*

Description

Methods for comparing neuroimaging objects

This method compares two NeuroVec objects (`e1` and `e2`) using a generic comparison function. The dimensions of both objects are checked for compatibility before performing the comparison.

Usage

```
## S4 method for signature 'SparseNeuroVol,numeric'
Compare(e1, e2)

## S4 method for signature 'numeric,SparseNeuroVol'
Compare(e1, e2)

## S4 method for signature 'NeuroVec,NeuroVec'
Compare(e1, e2)
```

Arguments

`e1`	A NeuroVec object to be compared.
`e2`	A NeuroVec object to be compared.

Value

The result of the comparison between the SparseNeuroVol object's data and the numeric value.

The result of the comparison between `e1` and `e2`.

`concat` *Concatenate two objects in the time dimension*

Description

Concatenate two objects in the time dimension

Usage

```
concat(x, y, ...)

## S4 method for signature 'NeuroVec,NeuroVol'
concat(x, y, ...)

## S4 method for signature 'NeuroVol,NeuroVec'
```

```
concat(x, y, ...)

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

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

## S4 method for signature 'DenseNeuroVol,missing'
concat(x, y, ...)

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

## S4 method for signature 'AbstractSparseNeuroVec,missing'
concat(x, y, ...)

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

Arguments

`x`	the first object, typically `NeuroVol` or `NeuroVec`
`y`	the second object, typically `NeuroVol` or `NeuroVec`
`...`	additional objects

Details

The `x` and `y` images must have compatible dimensions. A `NeuroVol` can be concatenated to `NeuroVec`, and vice versa. See examples.

Value

A temporally concatenated object.

Note

dimensions of x and y must be equal

Examples

```
bv1 <- NeuroVol(rep(1,1000), NeuroSpace(c(10,10,10), c(1,1,1)))
bv2 <- NeuroVol(rep(2,1000), NeuroSpace(c(10,10,10), c(1,1,1)))
bv3 <- concat(bv1,bv2)
inherits(bv3, "NeuroVec")

bv4 <- concat(bv3, bv1)
dim(bv4)[4] == 3
bv5 <- concat(bv1, bv3)
dim(bv4)[4] == 3
```

```
bv6 <- concat(bv4,bv5)
dim(bv6)[4] == 6
```

---

conn_comp *Connected components*

Description

Find connected components in an image. This function identifies and labels spatially connected regions in neuroimaging data, supporting both binary masks and thresholded volumes.

Usage

```
conn_comp(x, ...)

## S4 method for signature 'NeuroVol'
conn_comp(
  x,
  threshold = 0,
  cluster_table = TRUE,
  local_maxima = TRUE,
  local_maxima_dist = 15,
  ...
)
```

Arguments

`x`	the image object
`...`	additional arguments including: • threshold - numeric value defining lower intensity bound for image mask • cluster_table - logical indicating whether to return cluster statistics • local_maxima - logical indicating whether to compute local maxima • local_maxima_dist - minimum distance between local maxima • connect - connectivity pattern ("26-connect", "18-connect", or "6-connect")
`threshold`	threshold defining lower intensity bound for image mask
`cluster_table`	return cluster_table
`local_maxima`	return table of local maxima
`local_maxima_dist`	the distance used to define minum distance between local maxima

Value

A list containing:

- index - A `ClusteredNeuroVol` object with cluster labels
- size - A `NeuroVol` object with cluster sizes
- voxels - A list of cluster voxel coordinates
- cluster_table - (optional) Data frame with cluster statistics
- local_maxima - (optional) Matrix of local maxima coordinates

An object representing the connected components of `x`.

Examples

```
# Create a simple 3D volume with two distinct regions
space <- NeuroSpace(c(10,10,10), c(1,1,1))
vol_data <- array(0, c(10,10,10))

# Create first cluster in corner (2x2x2)
vol_data[1:2, 1:2, 1:2] <- 1

# Create second cluster in opposite corner (2x2x2)
vol_data[8:9, 8:9, 8:9] <- 1

# Create NeuroVol object
vol <- NeuroVol(vol_data, space)

# Find connected components with default 26-connectivity
# Returns components above threshold 0
comps <- conn_comp(vol, threshold=0)

# Access results
max(comps$index) == 2  # Should have 2 clusters
all(comps$size >= 0)    # All clusters should have >= 0

# Get cluster statistics
comps <- conn_comp(vol, threshold=0, cluster_table=TRUE)
# cluster_table contains: index, x, y, z, N (size), Area, value

# Find local maxima within clusters
comps <- conn_comp(vol, threshold=0, local_maxima=TRUE,
                  local_maxima_dist=2)
# local_maxima contains: index, x, y, z, value
```

`conn_comp_3D` *Extract Connected Components from a 3D Binary Mask*

Description

Identifies and labels connected components in a 3D binary mask using a two-pass algorithm. The function supports different connectivity constraints and returns both component indices and their sizes.

Usage

```
conn_comp_3D(mask, connect = c("26-connect", "18-connect", "6-connect"))
```

Arguments

`mask`	A 3D logical array representing the binary mask
`connect`	A character string specifying the connectivity constraint. One of "26-connect" (default), "18-connect", or "6-connect"

Details

The function implements an efficient two-pass connected component labeling algorithm:

- First pass: Assigns provisional labels and builds an equivalence table using a union-find data structure for label resolution
- Second pass: Resolves label conflicts and assigns final component labels

The connectivity options determine which voxels are considered adjacent:

- 6-connect: Only face-adjacent voxels (±1 step along each axis)
- 18-connect: Face and edge-adjacent voxels
- 26-connect: Face, edge, and vertex-adjacent voxels (all neighbors in a 3x3x3 cube)

Time complexity is O(n) where n is the number of voxels in the mask, with additional O(k) space for the union-find data structure where k is the number of provisional labels.

Value

A list with the following components:

`index`	A 3D array of integers. Each non-zero value represents the cluster index of the connected component for that voxel. Zero values indicate background.
`size`	A 3D array of integers. Each non-zero value represents the size (number of voxels) of the connected component that the voxel belongs to. Zero values indicate background.

References

Rosenfeld, A., & Pfaltz, J. L. (1966). Sequential operations in digital picture processing. Journal of the ACM, 13(4), 471-494.

See Also

`array` for creating 3D arrays, `ClusteredNeuroVol` for working with clustered neuroimaging data

Examples

```
# Create a simple 3D binary mask with two disconnected components
mask <- array(FALSE, c(4, 4, 4))
mask[1:2, 1:2, 1:2] <- TRUE  # First component
mask[3:4, 3:4, 3:4] <- TRUE  # Second component

# Extract components using different connectivity patterns
comps <- conn_comp_3D(mask, connect = "6-connect")

# Number of components
max_comps <- max(comps$index)
cat("Found", max_comps, "components\n")

# Size of each component
unique_sizes <- unique(comps$size[comps$size > 0])
cat("Component sizes:", paste(unique_sizes, collapse=", "), "\n")

# Try with different connectivity
comps_26 <- conn_comp_3D(mask, connect = "26-connect")
cat("Number of components with 26-connectivity:", max(comps_26$index), "\n")
```

`coords` *Extract coordinates from an object*

Description

This function extracts the coordinates from an input object.

Usage

```
coords(x, ...)
```

Arguments

`x`	The object to extract coordinates from.
`...`	Additional arguments (not used in the generic function).

Value

A numeric `matrix` or `vector` containing the coordinates of `x`.

Examples

```
# Create a NeuroSpace object with 3mm voxels
space <- NeuroSpace(c(10,10,10), spacing=c(3,3,3))

# Create ROI coordinates in voxel space
coords <- matrix(c(1,1,1, 2,2,2), ncol=3, byrow=TRUE)
roi_coords <- ROICoords(coords)

# Get coordinates in voxel space
vox_coords <- coords(roi_coords)
# First coordinate is (1,1,1)

# Get coordinates
cds <- coords(roi_coords)
nrow(cds) == 2
```

`coords,IndexLookupVol-method`

Extract Coordinates from an IndexLookupVol Object

Description

Extracts the coordinates from an IndexLookupVol object based on a given index.

Usage

```
## S4 method for signature 'IndexLookupVol'
coords(x, i)

## S4 method for signature 'ROIVol'
coords(x, real = FALSE)

## S4 method for signature 'ROICoords'
coords(x, real = FALSE)

## S4 method for signature 'AbstractSparseNeuroVec'
coords(x, i)
```

Arguments

`x`	An `IndexLookupVol` object to extract coordinates from
`i`	The index into the lookup volume
`real`	if `TRUE`, return coordinates in real world units

Value

A matrix of coordinates

Examples

```
space <- NeuroSpace(c(64, 64, 64), c(1, 1, 1), c(0, 0, 0))
ilv <- IndexLookupVol(space, c(1:100))
coords(ilv, 1)  # Extract coordinates for index 1
```

`coord_to_grid` *convert n-dimensional real world coordinates to grid coordinates*

Description

convert n-dimensional real world coordinates to grid coordinates

Usage

```
coord_to_grid(x, coords)

## S4 method for signature 'NeuroSpace,matrix'
coord_to_grid(x, coords)

## S4 method for signature 'NeuroSpace,numeric'
coord_to_grid(x, coords)

## S4 method for signature 'NeuroVol,matrix'
coord_to_grid(x, coords)

## S4 method for signature 'NeuroVol,numeric'
coord_to_grid(x, coords)
```

Arguments

`x`	the object
`coords`	a matrix of real world coordinates

Value

A numeric `matrix` of grid coordinates.

Examples

```
# Create a simple 3D volume
bvol <- NeuroVol(array(0, c(10,10,10)), NeuroSpace(c(10,10,10), c(1,1,1)))
coords <- matrix(c(.5,.5,.5, 1.5,1.5,1.5), ncol=3, byrow=TRUE)
grid <- coord_to_grid(bvol, coords)
world <- grid_to_coord(bvol, grid)
all.equal(coords, world)
```

`coord_to_index` *convert n-dimensional real world coordinates to 1D indices*

Description

convert n-dimensional real world coordinates to 1D indices

Usage

```
coord_to_index(x, coords)

## S4 method for signature 'NeuroSpace,matrix'
coord_to_index(x, coords)

## S4 method for signature 'NeuroSpace,numeric'
coord_to_index(x, coords)

## S4 method for signature 'NeuroVol,matrix'
coord_to_index(x, coords)
```

Arguments

`x`	the object
`coords`	a matrix of real world coordinates

Value

An integer `vector` of 1D indices corresponding to `coords`.

Examples

```
bvol <- NeuroVol(array(0, c(10,10,10)), NeuroSpace(c(10,10,10), c(1,1,1)))
coords <- matrix(c(.5,.5,.5, 1.5,1.5,1.5), ncol=3, byrow=TRUE)
idx <- coord_to_index(bvol, coords)
coords2 <- index_to_coord(bvol, idx)
all.equal(coords, coords2)
```

`createNIfTIHeader` *Create an Empty NIfTI-1 Header List*

Description

Initializes a list of fields following the NIfTI-1 specification with default or placeholder values. Users typically call this internally via `as_nifti_header` rather than using directly.

Usage

```
createNIfTIHeader(oneFile = TRUE, file_name = NULL)
```

Arguments

`oneFile`	Logical; if `TRUE`, `magic` is set to `"n+1"` indicating a single-file (.nii) approach. Otherwise set to `"ni1"`.
`file_name`	Optional character string to store in the header, usually referencing the intended output file name.

Details

This function sets up the skeleton of a NIfTI-1 header, including fields for `diminfo`, `pixdim`, `qform_code`, `magic`, etc. Most fields are initialized to zero, empty characters, or standard placeholders. The `oneFile` argument controls whether `"n+1"` or `"ni1"` is used for the `magic` field.

Value

A named `list` containing approximately 30 fields that comprise the NIfTI-1 header structure. Many of these are placeholders until filled by downstream usage.

See Also

`as_nifti_header` for populating the returned header with actual data from a `NeuroVol`.

`cuboid_roi` *Create A Cuboid Region of Interest*

Description

Create A Cuboid Region of Interest

Usage

```
cuboid_roi(bvol, centroid, surround, fill = NULL, nonzero = FALSE)
```

Arguments

`bvol`	an `NeuroVol` or `NeuroSpace` instance
`centroid`	the center of the cube in *voxel* coordinates
`surround`	the number of voxels on either side of the central voxel. A `vector` of length 3.
`fill`	optional value(s) to assign to data slot.
`nonzero`	keep only nonzero elements from `bvol`. If `bvol` is A `NeuroSpace` then this argument is ignored.

Value

An instance of class `ROIVol` representing the cuboid region of interest, containing the coordinates and values of voxels within the specified region.

Examples

```
sp1 <- NeuroSpace(c(10,10,10), c(1,1,1))
cube <- cuboid_roi(sp1, c(5,5,5), 3)
vox <- coords(cube)
cube2 <- cuboid_roi(sp1, c(5,5,5), 3, fill=5)
```

`data_file` *Generic function to get the name of the data file, given a file name and a `FileFormat` instance.*

Description

Derives the data file name from a given file name based on the FileFormat specifications.

Usage

```
data_file(x, file_name)

## S4 method for signature 'FileFormat,character'
data_file(x, file_name)
```

Arguments

`x`	A FileFormat object specifying the format requirements
`file_name`	A character string specifying the file name to derive the data file name from

Details

The function performs the following steps:

1. If the input file_name already matches the data file format, it returns the file_name as is.
2. If the file_name matches the header file format, it constructs and returns the corresponding data file name.
3. If the file_name doesn't match either format, it throws an error.

Value

The correct data file name as a `character` string.

A character string representing the data file name

See Also

`header_file`, `strip_extension` for related file name manipulation

Examples

```
fmt <- new("FileFormat", header_extension = "hdr", data_extension = "img")
data_file(fmt, "brain_scan.img")  # Returns "brain_scan.img"
data_file(fmt, "brain_scan.hdr")  # Also Returns "brain_scan.img"
```

`data_file_matches` *Generic function to test whether a file name conforms to the given a `FileFormat` instance. Will test for match to data file only*

Description

Validates whether a file name conforms to the data file format specification.

Usage

```
data_file_matches(x, file_name)

## S4 method for signature 'FileFormat,character'
data_file_matches(x, file_name)
```

Arguments

`x`	A FileFormat object specifying the format requirements
`file_name`	A character string specifying the file name to validate

Details

The function performs case-sensitive pattern matching to verify that the file name ends with the specified data extension. The match is performed using a regular expression that ensures the extension appears at the end of the file name.

Value

`TRUE` for match, `FALSE` otherwise.

See Also

`file_matches`, `header_file_matches` for related file format validation

Examples

```
fmt <- new("FileFormat", header_extension = "hdr", data_extension = "img")
data_file_matches(fmt, "brain_scan.img")  # TRUE
data_file_matches(fmt, "brain_scan.hdr")  # FALSE
data_file_matches(fmt, "brain.img.gz")    # FALSE
```

`data_reader` *Create a Data Reader*

Description

Creates a data reader for accessing neuroimaging data from various file formats. The reader provides a unified interface for reading data regardless of the underlying format.

Usage

```
data_reader(x, offset)
```

Arguments

`x`	An object containing metadata required to create the reader (e.g., file path, format info)
`offset`	Numeric. Byte offset where data reading should begin. Default is 0.

Details

Create a Data Reader for Neuroimaging Data

The data_reader function is a generic that creates appropriate readers for different neuroimaging formats. It handles:

- File format detection and validation

- Endianness configuration
- Data type conversion
- Compression handling (e.g., gzip)
- Proper byte alignment

Value

A BinaryReader object configured for the specific data format

See Also

`read_header` for reading headers, `BinaryReader` for reading binary data

Examples

```
# Create reader for NIFTI file
meta <- read_header(system.file("extdata", "global_mask_v4.nii", package="neuroim2"))
reader <- data_reader(meta, offset = 0)

# Read first 100 voxels
data <- read_elements(reader, 100)
```

`data_reader,NIFTIMetaInfo-method`

Create Data Reader for AFNI Format

Description

Create Data Reader for AFNI Format

Usage

```
## S4 method for signature 'NIFTIMetaInfo'
data_reader(x, offset = 0)

## S4 method for signature 'AFNIMetaInfo'
data_reader(x, offset = 0)
```

Arguments

`x`	AFNIMetaInfo object
`offset`	Numeric byte offset

Value

BinaryReader object

`DenseNeuroVec-class` *DenseNeuroVec Class*

Description

A class representing a four-dimensional brain image, backed by a dense array. This class is designed for neuroimaging data where most voxels contain non-zero values.

This function constructs a DenseNeuroVec object, which represents a dense four-dimensional brain image. It handles various input data formats and ensures proper dimensionality.

Usage

```
DenseNeuroVec(data, space, label = "none")
```

Arguments

`data`	The image data. This can be: • A 4-dimensional array • A 2-dimensional matrix (either nvoxels x ntime-points or ntime-points x nvoxels) • A vector (which will be reshaped to match the space dimensions)
`space`	A `NeuroSpace` object defining the spatial properties of the image.
`label`	A character string providing a label for the DenseNeuroVec object. Default is an empty string.

Details

DenseNeuroVec objects store their data in a dense array format, which is efficient for operations that require frequent access to all voxels. This class inherits from both `NeuroVec` and `array` classes, combining spatial information with array-based storage.

The function performs several operations based on the input data type:

- For matrix input: It determines the correct orientation (voxels x time or time x voxels) and reshapes accordingly. If necessary, it adds a 4th dimension to the space object.
- For vector input: It reshapes the data to match the dimensions specified in the space object.
- For array input: It ensures the dimensions match those specified in the space object.

Note that the label parameter is currently not used in the object creation, but is included for potential future use or consistency with other constructors.

Value

A concrete instance of the `DenseNeuroVec` class.

Validity

A DenseNeuroVec object is considered valid if:

- The underlying data is a four-dimensional array.

See Also

`NeuroVec-class` for the parent class. `SparseNeuroVec-class` for a sparse representation alternative.

`NeuroVec-class` for the parent class. `SparseNeuroVec-class` for the sparse version of 4D brain images. `NeuroSpace-class` for details on spatial properties.

Examples

```
# Create a simple 4D brain image
data <- array(rnorm(64*64*32*10), dim = c(64, 64, 32, 10))
space <- NeuroSpace(dim = c(64, 64, 32,10), origin = c(0, 0, 0), spacing = c(3, 3, 4))
dense_vec <- new("DenseNeuroVec", .Data = data, space = space)

# Access dimensions
dim(dense_vec)

# Extract a single 3D volume
first_volume <- dense_vec[[1]]


# Create a simple 4D brain image
dim <- c(64, 64, 32, 10)  # 64x64x32 volume with 10 time points
data <- array(rnorm(prod(dim)), dim)
space <- NeuroSpace(dim, spacing = c(3, 3, 4))

# Create a DenseNeuroVec object
dense_vec <- DenseNeuroVec(data = data, space = space, label = "Example")
print(dense_vec)

# Create from a matrix (voxels x time)
mat_data <- matrix(rnorm(prod(dim)), nrow = prod(dim[1:3]), ncol = dim[4])
dense_vec_mat <- DenseNeuroVec(data = mat_data, space = space)
print(dense_vec_mat)
```

`DenseNeuroVol-class` *DenseNeuroVol Class*

Description

Represents a three-dimensional brain image backed by a dense array. This class combines the spatial properties of `NeuroVol` with the data storage capabilities of an array.

Construct a `DenseNeuroVol` instance

Usage

```
DenseNeuroVol(data, space, label = "", indices = NULL)
```

Arguments

`data`	a three-dimensional `array`
`space`	an instance of class `NeuroSpace`
`label`	a `character` string
`indices`	an optional 1-d index vector

Details

DenseNeuroVol objects are used for 3D brain images where most or all voxels contain meaningful data. They provide efficient access to individual voxel values and are suitable for operations that require frequent random access to voxel data.

Value

`DenseNeuroVol` instance

See Also

`NeuroVol-class`, `SparseNeuroVol-class`

Examples

```
# Create a simple 3D brain volume
vol_data <- array(rnorm(64*64*64), c(64, 64, 64))
vol_space <- NeuroSpace(dim=c(64L, 64L, 64L), origin=c(0, 0, 0), spacing=c(1, 1, 1))
brain_vol <- new("DenseNeuroVol", .Data=vol_data, space=vol_space)
```

`dim,ClusteredNeuroVec-method`
Get Dimensions of FileMetaInfo Object

Description

Get Dimensions of FileMetaInfo Object

dim of `NeuroObj` object

Usage

```
## S4 method for signature 'ClusteredNeuroVec'
dim(x)

## S4 method for signature 'FileMetaInfo'
dim(x)

## S4 method for signature 'NeuroObj'
dim(x)

## S4 method for signature 'NeuroSpace'
dim(x)

## S4 method for signature 'ROIVol'
dim(x)

## S4 method for signature 'ROICoords'
dim(x)
```

Arguments

`x`	the object

Value

A numeric vector of length 2 containing the dimensions of the ROICoords object.

`dim_of` *Get the length of a given dimension of an object*

Description

This function returns the length of a given axis (dimension) of an object. The axis can be specified using its position or name.

Usage

```
dim_of(x, axis)

## S4 method for signature 'NeuroSpace,NamedAxis'
dim_of(x, axis)
```

Arguments

`x`	The NeuroSpace object
`axis`	The NamedAxis to query

Value

An integer representing the length of the specified `axis` of `x`.

Examples

```
x <- NeuroSpace(c(10,10,10), spacing=c(1,1,1))
stopifnot(dim_of(x, x@axes@i) == 10)
```

`downsample` *Downsample an Image*

Description

This function downsamples a neuroimaging object, reducing its spatial resolution while preserving the temporal dimension.

Usage

```
downsample(x, ...)

## S4 method for signature 'DenseNeuroVec'
downsample(x, spacing = NULL, factor = NULL, outdim = NULL, method = "box")

## S4 method for signature 'NeuroVec'
downsample(x, spacing = NULL, factor = NULL, outdim = NULL, method = "box")

## S4 method for signature 'DenseNeuroVol'
downsample(x, spacing = NULL, factor = NULL, outdim = NULL, method = "box")

## S4 method for signature 'NeuroVol'
downsample(x, spacing = NULL, factor = NULL, outdim = NULL, method = "box")
```

Arguments

`x`	A DenseNeuroVol object to downsample
`...`	Additional arguments passed to specific downsample methods.
`spacing`	Target voxel spacing (numeric vector of length 3)
`factor`	Downsampling factor (single value or vector of length 3, between 0 and 1)
`outdim`	Target output dimensions (numeric vector of length 3)
`method`	Downsampling method (currently only "box" for box averaging)

Value

An object of the same class as `x`, downsampled according to the specified parameters.

Examples

```
# Create a sample 4D image
data <- array(rnorm(64*64*32*10), dim = c(64, 64, 32, 10))
space <- NeuroSpace(dim = c(64, 64, 32, 10),
                    origin = c(0, 0, 0),
                    spacing = c(2, 2, 2))
nvec <- DenseNeuroVec(data, space)

# Downsample by factor
nvec_down1 <- downsample(nvec, factor = 0.5)

# Downsample to target spacing
nvec_down2 <- downsample(nvec, spacing = c(4, 4, 4))

# Downsample to target dimensions
nvec_down3 <- downsample(nvec, outdim = c(32, 32, 16))

# Create a sample 3D volume
data <- array(rnorm(64*64*32), dim = c(64, 64, 32))
space <- NeuroSpace(dim = c(64, 64, 32),
                    origin = c(0, 0, 0),
                    spacing = c(2, 2, 2))
vol <- DenseNeuroVol(data, space)

# Downsample by factor
vol_down1 <- downsample(vol, factor = 0.5)

# Downsample to target spacing
vol_down2 <- downsample(vol, spacing = c(4, 4, 4))

# Downsample to target dimensions
vol_down3 <- downsample(vol, outdim = c(32, 32, 16))
```

`drop` *Generic Drop Method*

Description

Provides a mechanism to remove dimensions or elements from an object.

Usage

```
drop(x)
```

Arguments

`x`	An object.

Value

An object of the same class as `x` with reduced dimensions or elements.

`drop,NeuroVec-method` *Drop a dimension*

Description

Drop a dimension

Usage

```
## S4 method for signature 'NeuroVec'
drop(x)
```

Arguments

`x`	the object to drop a dimension from

Value

An object of the same class as x with reduced dimensions or elements.

`drop_dim` *Drop a Dimension from an Object*

Description

This function removes a specified dimension from a given object, such as a matrix or an array.

Usage

```
drop_dim(x, dimnum)

## S4 method for signature 'AxisSet2D,numeric'
drop_dim(x, dimnum)

## S4 method for signature 'AxisSet2D,missing'
drop_dim(x, dimnum)

## S4 method for signature 'AxisSet3D,numeric'
drop_dim(x, dimnum)

## S4 method for signature 'AxisSet3D,missing'
drop_dim(x, dimnum)
```

```
## S4 method for signature 'NeuroSpace,numeric'
drop_dim(x, dimnum)

## S4 method for signature 'NeuroSpace,missing'
drop_dim(x)
```

Arguments

`x`	An AxisSet3D object
`dimnum`	Numeric index of dimension to drop (optional)

Value

An object of the same class as `x` with the specified dimension removed.

Examples

```
# Create a NeuroSpace object with dimensions (10, 10, 10)
x <- NeuroSpace(c(10, 10, 10), c(1, 1, 1))

# Drop the first dimension
x1 <- drop_dim(x, 1)

# Check the new dimensions
ndim(x1) == 2
dim(x1)[1] == 10
```

`ecode_name` *Get Extension Code Name*

Description

Returns the name associated with a NIfTI extension code.

Usage

```
ecode_name(ecode)
```

Arguments

`ecode`	Integer extension code.

Value

Character string with the extension name, or "unknown" if not found.

Examples

```
ecode_name(4L)   # Returns "AFNI"
ecode_name(6L)   # Returns "comment"
ecode_name(999L) # Returns "unknown"
```

`embed_kernel` *Generic function to position kernel in a position in image space*

Description

Generic function to position kernel in a position in image space

Usage

```
embed_kernel(x, sp, center_voxel, ...)

## S4 method for signature 'Kernel,NeuroSpace,numeric'
embed_kernel(x, sp, center_voxel, weight = 1)
```

Arguments

`x`	the kernel object
`sp`	the space to embed the kernel
`center_voxel`	the voxel marking the center of the kernel in the embedded space
`...`	extra args
`weight`	multiply kernel weights by this value

Value

An object representing the embedded kernel in the specified space.

Examples

```
# Create a 3D Gaussian kernel with dimensions 3x3x3 and voxel size 1x1x1
kern <- Kernel(kerndim = c(3,3,3), vdim = c(1,1,1), FUN = dnorm, sd = 1)

# Create a NeuroSpace object to embed the kernel in
space <- NeuroSpace(c(10,10,10), c(1,1,1))

# Embed the kernel at the center of the space (position 5,5,5)
embedded_kern <- embed_kernel(kern, space, c(5,5,5))

# The result is a SparseNeuroVol with kernel weights centered at (5,5,5)
# We can also scale the kernel weights by using the weight parameter
embedded_kern_scaled <- embed_kernel(kern, space, c(5,5,5), weight = 2)
```

```
# The scaled kernel has weights twice as large as the original
max(values(embedded_kern_scaled)) == 2 * max(values(embedded_kern))
```

`extension` *Get Extension by Code*

Description

Retrieve extensions with a specific extension code from a list.

Usage

```
extension(x, ecode)

## S4 method for signature 'NiftiExtensionList,numeric'
extension(x, ecode)
```

Arguments

`x`	A `NiftiExtensionList-class` object.
`ecode`	Integer extension code to filter by.

Value

A `NiftiExtensionList-class` containing only extensions with the specified code.

`extensions` *Get Extensions from an Object*

Description

Generic function to retrieve NIfTI extensions from various object types.

Usage

```
extensions(x, ...)
```

Arguments

`x`	An object potentially containing extensions.
`...`	Additional arguments (currently unused).

Value

A `NiftiExtensionList-class` object, or NULL if no extensions.

`extractor3d` *Array-like access for 3-dimensional data structures*

Description

This generic function provides array-like access for 3-dimensional data structures. It allows for flexible indexing and subsetting of 3D arrays or array-like objects.

Usage

```
## S4 method for signature 'ArrayLike3D,numeric,missing,ANY'
x[i, j, k, ..., drop = TRUE]

## S4 method for signature 'ArrayLike3D,matrix,missing,ANY'
x[i, j, k, ..., drop = TRUE]

## S4 method for signature 'ArrayLike3D,missing,missing,ANY'
x[i, j, k, ..., drop = TRUE]

## S4 method for signature 'ArrayLike3D,missing,numeric,ANY'
x[i, j, k, ..., drop = TRUE]
```

Arguments

`x`	The 3-dimensional object to be accessed.
`i`	First index or dimension.
`j`	Second index or dimension.
`k`	Third index or dimension.
`...`	Additional arguments passed to methods.
`drop`	Logical. If TRUE, the result is coerced to the lowest possible dimension.

Value

A subset of the input object, with dimensions depending on the indexing and the ‘drop‘ parameter.

`extractor4d` *Array-like access for 4-dimensional data structures*

Description

This generic function provides array-like access for 4-dimensional data structures. It allows for flexible indexing and subsetting of 4D arrays or array-like objects.

Provides array-like access to ClusteredNeuroVec objects, supporting extraction patterns like x[,,,t] to get 3D volumes at specific time points.

Usage

```
## S4 method for signature 'ArrayLike4D,matrix,missing,ANY'
x[i, j, k, m, ..., drop = TRUE]

## S4 method for signature 'ArrayLike4D,numeric,numeric,ANY'
x[i, j, k, m, ..., drop = TRUE]

## S4 method for signature 'ArrayLike4D,numeric,missing,ANY'
x[i, j, k, m, ..., drop = TRUE]

## S4 method for signature 'ArrayLike4D,integer,missing,ANY'
x[i, j, k, m, ..., drop = TRUE]

## S4 method for signature 'ArrayLike4D,missing,missing,ANY'
x[i, j, k, m, ..., drop = TRUE]

## S4 method for signature 'ArrayLike4D,missing,numeric,ANY'
x[i, j, k, m, ..., drop = TRUE]

## S4 method for signature 'ClusteredNeuroVec,missing,missing,ANY'
x[i, j, k, m, ..., drop = TRUE]

## S4 method for signature 'ClusteredNeuroVec,missing,missing,ANY'
x[i, j, k, m, ..., drop = TRUE]

## S4 method for signature 'ClusteredNeuroVec,numeric,numeric,ANY'
x[i, j, k, m, ..., drop = TRUE]
```

Arguments

`x`	The 4-dimensional object to be accessed.
`i`	First index or dimension.
`j`	Second index or dimension.
`k`	Third index or dimension.
`m`	Fourth index or dimension.
`...`	Additional arguments passed to methods.
`drop`	Logical. If TRUE, the result is coerced to the lowest possible dimension.

Value

A subset of the input object, with dimensions depending on the indexing and the ‘drop‘ parameter.

`FileBackedNeuroVec` *Create a File-Backed Neuroimaging Vector*

Description

Constructs a `FileBackedNeuroVec` instance, which represents a file-backed neuroimaging vector object. This constructor provides memory-efficient access to large neuroimaging datasets by keeping the data on disk until needed.

Usage

```
FileBackedNeuroVec(file_name, label = basename(file_name))
```

Arguments

`file_name`	A character string specifying the path to the neuroimaging file. Supported formats include NIFTI (.nii) and ANALYZE (.hdr/.img).
`label`	Optional character string providing a label for the vector

Details

Create a FileBackedNeuroVec Object

The function performs the following operations:

- Reads the header information from the specified file
- Validates the dimensionality (must be 4D data)
- Creates a `NeuroSpace` object with appropriate metadata
- Initializes the file-backed vector with minimal memory footprint

Value

A new instance of class `FileBackedNeuroVec`.

See Also

`NeuroSpace` for spatial metadata management, `read_header` for header information extraction, `sub_vector` for data access methods

Examples

```
# Create a file-backed vector from a NIFTI file
fbvec <- FileBackedNeuroVec(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))

# Access specific volumes without loading entire dataset
first_vol <- sub_vector(fbvec, 1)
```

`FileBackedNeuroVec-class`

FileBackedNeuroVec Class

Description

A class representing a four-dimensional brain image that uses on-demand loading through memory-mapped file access. This approach enables efficient handling of large-scale brain imaging data by loading only the required portions of the data into memory when needed.

The `FileBackedNeuroVec` class represents a memory-efficient vector of neuroimaging data that is stored on disk rather than in memory. This is particularly useful for large datasets where memory constraints are a concern.

Details

FileBackedNeuroVec objects provide a memory-efficient solution for working with large 4D neuroimaging datasets. By utilizing memory-mapped file access, this class allows users to work with datasets that exceed available RAM, only loading the necessary data segments into memory as they are accessed.

Slots

`meta` An instance of class `FileMetaInfo` containing file metadata such as file path, format, and other associated information.

Inheritance

`FileBackedNeuroVec` inherits from:

- `NeuroVec`: Base class for 4D brain images
- `ArrayLike4D`: Interface for 4D array-like operations

Memory Management

Data is read from disk on-demand, reducing memory usage compared to in-memory storage. The trade-off is slightly slower access times due to disk I/O operations.

See Also

`NeuroVec-class` for the base 4D brain image class. `FileMetaInfo-class` for details on file metadata representation.

`FileBackedNeuroVec` for creating instances of this class

Examples

```
# Load example 4D image file included with package
file_path <- system.file("extdata", "global_mask_v4.nii", package = "neuroim2")
fbvec <- FileBackedNeuroVec(file_path)

# Get dimensions of the image
dim(fbvec)

# Extract first volume
vol1 <- sub_vector(fbvec, 1)

# Extract multiple volumes
vols <- sub_vector(fbvec, 1:2)
```

`FileFormat-class` *FileFormat Class*

Description

This class represents a neuroimaging file format descriptor, containing information about the file format, encoding, and extensions for both header and data components.

Slots

`file_format` A `character` string specifying the name of the file format (e.g., "NIfTI").

`header_encoding` A `character` string specifying the file encoding of the header file (e.g., "raw" for binary, "gzip" for gz compressed).

`header_extension` A `character` string specifying the file extension for the header file (e.g., "nii" for NIfTI single files).

`data_encoding` A `character` string specifying the file encoding for the data file.

`data_extension` A `character` string specifying the file extension for the data file (e.g., "nii" for NIfTI single files).

Examples

```
# Create a FileFormat object for NIfTI format
nifti_format <- new("FileFormat",
                    file_format = "NIfTI",
                    header_encoding = "raw",
                    header_extension = "nii",
                    data_encoding = "raw",
                    data_extension = "nii")
```

`FileFormat-operations` *File Format Operations for Neuroimaging Data*

Description

A collection of methods for handling neuroimaging file formats with separate header and data files (e.g., ANALYZE, NIFTI). These methods provide functionality for file name validation, extension handling, and file path manipulation.

File Format Structure

Neuroimaging formats often use paired files:

- A header file (e.g., '.hdr') containing metadata
- A data file (e.g., '.img') containing the actual image data

Common Operations

- Validating file names against format specifications
- Converting between header and data file names
- Checking file existence and compatibility

`FileMetaInfo-class` *FileMetaInfo Class*

Description

This class extends MetaInfo to include file-specific metadata for neuroimaging data files.

This class extends FileMetaInfo with NIfTI-specific metadata.

This class extends FileMetaInfo with AFNI-specific metadata.

Slots

`header_file` A `character` string specifying the name of the file containing meta information.

`data_file` A `character` string specifying the name of the file containing image data.

`descriptor` A `FileFormat` object describing the image file format.

`endian` A `character` string specifying the byte order of data ('little' or 'big').

`data_offset` A `numeric` value indicating the number of bytes preceding the start of image data in the data file.

`bytes_per_element` An `integer` specifying the number of bytes per data element.

`intercept` A `numeric` vector of constant values added to image data (one per sub-image).

`slope` A `numeric` vector of multipliers for image data (one per sub-image).

`header` A `list` of format-specific attributes.

`nifti_header` A `list` of attributes specific to the NIfTI file format.

`afni_header` A `list` of attributes specific to the AFNI file format.

See Also

`MetaInfo-class`, `NIFTIMetaInfo-class`, `AFNIMetaInfo-class`

`FileMetaInfo-class`

`FileMetaInfo-class`

`FileSource-class` *FileSource Class*

Description

Base class for representing a data source for images. The purpose of this class is to provide a layer in between low level IO and image loading functionality.

Slots

`meta_info` An object of class `FileMetaInfo` containing meta information for the data source.

`file_matches` *Generic function to test whether a file name conforms to the given `FileFormat` instance. Will test for match to either header file or data file*

Description

Validates whether a file name conforms to the specified FileFormat and verifies the existence of both header and data files.

Usage

```
file_matches(x, file_name)

## S4 method for signature 'FileFormat,character'
file_matches(x, file_name)
```

Arguments

`x`	A FileFormat object specifying the format requirements
`file_name`	A character string specifying the file name to validate

Details

The function performs the following validation steps:

1. Checks if the file name matches either the header or data format
2. Verifies the existence of the corresponding paired file
3. Returns `FALSE` if either check fails

File names are validated using case-sensitive extension matching.

Value

`TRUE` for match, `FALSE` otherwise.

A logical value: `TRUE` if the file matches the format and both header and data files exist, `FALSE` otherwise

See Also

`header_file_matches`, `data_file_matches` for individual file type checking

Examples

```
# Create a FileFormat for NIFTI format

fmt <- new("FileFormat",
  file_format = "NIFTI",
  header_encoding = "raw",
  header_extension = "nii",
  data_encoding = "raw",
  data_extension = "nii")

# Create temporary file
tmp <- tempfile("brainscan", fileext = ".nii")
file.create(tmp)

# Check if files exist and match format
file_matches(fmt, tmp)

# Clean up
unlink(tmp)
```

`findAnatomy3D` *Find 3D anatomical orientation from axis abbreviations*

Description

Creates a 3D anatomical orientation from axis abbreviations.

Usage

```
findAnatomy3D(axis1 = "L", axis2 = "P", axis3 = "I")
```

Arguments

`axis1`	Character string for first axis (default: "L" for Left)
`axis2`	Character string for second axis (default: "P" for Posterior)
`axis3`	Character string for third axis (default: "I" for Inferior)

Value

An AxisSet3D object representing the anatomical orientation

Examples

```
# Create orientation with default LPI axes
orient <- findAnatomy3D()
# Create orientation with custom axes
orient <- findAnatomy3D("R", "A", "S")
```

`gaussian_blur` *Gaussian Blur for Volumetric Images*

Description

This function applies an isotropic discrete Gaussian kernel to smooth a volumetric image (3D brain MRI data). The blurring is performed within a specified image mask, with customizable kernel parameters.

Usage

```
gaussian_blur(vol, mask, sigma = 2, window = 1)
```

Arguments

`vol`	A `NeuroVol` object representing the image volume to be smoothed.
`mask`	An optional `LogicalNeuroVol` object representing the image mask. This mask defines the region where the blurring is applied. If not provided, the entire volume is processed.
`sigma`	A numeric value specifying the standard deviation of the Gaussian kernel. Default is 2.
`window`	An integer specifying the kernel size. It represents the number of voxels to include on each side of the center voxel. For example, window=1 results in a 3x3x3 kernel. Default is 1.

Details

The function uses a C++ implementation for efficient Gaussian blurring. The blurring is applied only to voxels within the specified mask (or the entire volume if no mask is provided). The kernel size is determined by the 'window' parameter, and its shape by the 'sigma' parameter.

Value

A `NeuroVol` object representing the smoothed image.

References

Gaussian blur: https://en.wikipedia.org/wiki/Gaussian_blur

See Also

`NeuroVol-class`, `LogicalNeuroVol-class`, `bilateral_filter`

Examples

```
# Load a sample brain mask
brain_mask <- read_vol(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))

# Apply Gaussian blurring to the brain volume
blurred_vol <- gaussian_blur(brain_mask, brain_mask, sigma = 2, window = 1)

# View a slice of the original and blurred volumes
image(brain_mask[,,12])
image(blurred_vol[,,12])
```

`get_afni_attribute` *Get AFNI Attribute from Extension*

Description

Extracts a specific attribute value from a parsed AFNI extension.

Usage

```
get_afni_attribute(ext, attr_name)
```

Arguments

`ext`	A `NiftiExtension-class` object with ecode = 4, or an xml_document from `parse_afni_extension`.
`attr_name`	Character string specifying the attribute name to retrieve (e.g., "HISTORY_NOTE", "BRICK_LABS").

Value

The attribute value, or NULL if not found. The type depends on the attribute's ni_type in the XML.

Examples

```
## Not run:
# Get the history note from an AFNI extension
history <- get_afni_attribute(afni_ext, "HISTORY_NOTE")

## End(Not run)
```

`grid_to_coord` *Generic function to convert N-dimensional grid coordinates to real world coordinates*

Description

Generic function to convert N-dimensional grid coordinates to real world coordinates

Usage

```
grid_to_coord(x, coords)

## S4 method for signature 'NeuroSpace,matrix'
grid_to_coord(x, coords)

## S4 method for signature 'NeuroSpace,matrix'
grid_to_coord(x, coords)

## S4 method for signature 'NeuroSpace,numeric'
grid_to_coord(x, coords)

## S4 method for signature 'NeuroVol,matrix'
grid_to_coord(x, coords)
```

Arguments

`x`	the object
`coords`	a matrix of grid coordinates

Value

A numeric `matrix` of real-world coordinates.

Examples

```
# Create a simple 3D volume
bvol <- NeuroVol(array(0, c(10,10,10)), NeuroSpace(c(10,10,10), c(1,1,1)))
grid_coords <- matrix(c(1.5,1.5,1.5, 5.5,5.5,5.5), ncol=3, byrow=TRUE)
world <- grid_to_coord(bvol, grid_coords)
grid <- coord_to_grid(bvol, world)
all.equal(grid_coords, grid)
```

`grid_to_grid` *Generic function to convert voxel coordinates in the reference space (LPI) to native array space.*

Description

Generic function to convert voxel coordinates in the reference space (LPI) to native array space.

Usage

```
grid_to_grid(x, vox)

## S4 method for signature 'NeuroSpace,matrix'
grid_to_grid(x, vox)

## S4 method for signature 'matrix,matrix'
grid_to_grid(x, vox)
```

Arguments

`x`	the object
`vox`	a matrix of LPI voxel coordinates

Value

A numeric `matrix` of native voxel coordinates.

Examples

```
# Create a simple 3D volume in LPI orientation
space <- NeuroSpace(c(10,10,10), c(2,2,2))

# Create a reoriented space in RAS orientation
space_ras <- reorient(space, c("R", "A", "S"))

# Convert coordinates between orientations
voxel_coords <- t(matrix(c(1,1,1)))
new_coords <- grid_to_grid(space_ras, voxel_coords)
print(new_coords)
```

`grid_to_index` *Generic function to convert N-dimensional grid coordinates to 1D indices*

Description

Converts 2D grid coordinates to linear indices for a `NeuroSlice` object.

Usage

```
grid_to_index(x, coords)

## S4 method for signature 'NeuroSlice,matrix'
grid_to_index(x, coords)

## S4 method for signature 'NeuroSlice,numeric'
grid_to_index(x, coords)

## S4 method for signature 'NeuroSpace,matrix'
grid_to_index(x, coords)

## S4 method for signature 'NeuroSpace,numeric'
grid_to_index(x, coords)

## S4 method for signature 'NeuroVol,matrix'
grid_to_index(x, coords)

## S4 method for signature 'NeuroVol,numeric'
grid_to_index(x, coords)
```

Arguments

`x`	A `NeuroSlice` object
`coords`	Either a numeric vector of length 2 or a matrix with 2 columns, representing (x,y) coordinates in the slice grid

Details

Convert Grid Coordinates to Linear Indices

Value

An integer `vector` of 1D indices corresponding to `coords`.

See Also

`index_to_grid` for the inverse operation

Examples

```
# Create a 2D space (10x10)
space_2d <- NeuroSpace(c(10,10), c(1,1))

# Convert 2D grid coordinates to linear indices
coords_2d <- matrix(c(1,1, 2,2), ncol=2, byrow=TRUE)
idx_2d <- grid_to_index(space_2d, coords_2d)
# First coordinate (1,1) maps to index 1
# Second coordinate (2,2) maps to index 12 (= 2 + (2-1)*10)

# Create a 3D space (10x10x10)
space_3d <- NeuroSpace(c(10,10,10), c(1,1,1))

# Convert 3D grid coordinates to linear indices
coords_3d <- matrix(c(1,1,1, 2,2,2), ncol=3, byrow=TRUE)
idx_3d <- grid_to_index(space_3d, coords_3d)

# Single coordinate can also be converted
idx <- grid_to_index(space_3d, c(1,1,1))

slice_space <- NeuroSpace(c(10, 10))
slice_data <- matrix(1:100, 10, 10)
slice <- NeuroSlice(slice_data, slice_space)

# Convert single coordinate
idx <- grid_to_index(slice, c(5, 5))

# Convert multiple coordinates
coords <- matrix(c(1,1, 2,2, 3,3), ncol=2, byrow=TRUE)
indices <- grid_to_index(slice, coords)
```

`guided_filter` *Edge-Preserving Guided Filter for Volumetric Images*

Description

This function applies a guided filter to a volumetric image (3D brain MRI data) to perform edge-preserving smoothing. The guided filter smooths the image while preserving edges, providing a balance between noise reduction and structural preservation.

Usage

```
guided_filter(vol, radius = 4, epsilon = 0.7^2)
```

Arguments

`vol`	A `NeuroVol` object representing the image volume to be filtered.
`radius`	An integer specifying the spatial radius of the filter. Default is 4.

`epsilon`	A numeric value specifying the regularization parameter. It controls the degree of smoothing and edge preservation. Default is 0.49 (0.7^2).

Details

The guided filter operates by computing local linear models between the guidance image (which is the same as the input image in this implementation) and the output. The 'radius' parameter determines the size of the local neighborhood, while 'epsilon' controls the smoothness of the filter.

The implementation uses box blur operations for efficiency, which approximates the behavior of the original guided filter algorithm.

Value

A `NeuroVol` object representing the filtered image.

References

He, K., Sun, J., & Tang, X. (2013). Guided Image Filtering. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(6), 1397-1409.

See Also

`gaussian_blur`, `bilateral_filter`, `NeuroVol-class`

Examples

```
# Load an example brain volume
brain_vol <- read_vol(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))

# Apply guided filtering to the brain volume

filtered_vol <- guided_filter(brain_vol, radius = 4, epsilon = 0.49)

# Visualize a slice of the original and filtered volumes
oldpar <- par(mfrow = c(1, 2))
image(brain_vol[,,12], main = "Original")
image(filtered_vol[,,12], main = "Filtered")
par(oldpar)
```

`has_extensions` *Check if Extensions are Present*

Description

Tests whether an object has any NIfTI extensions.

Usage

```
has_extensions(x)

## S4 method for signature 'NiftiExtensionList'
has_extensions(x)

## S4 method for signature 'list'
has_extensions(x)
```

Arguments

`x`	An object to test.

Value

Logical indicating whether extensions are present.

`header_file` *Generic function to get the name of the header file, given a file name and a `FileFormat` instance.*

Description

Derives the header file name from a given file name based on the FileFormat specifications.

Usage

```
header_file(x, file_name)

## S4 method for signature 'FileFormat,character'
header_file(x, file_name)
```

Arguments

`x`	A FileFormat object specifying the format requirements
`file_name`	A character string specifying the file name to derive the header file name from

Details

The function performs the following steps:

1. If the input file_name already matches the header file format, it returns the file_name as is.
2. If the file_name matches the data file format, it constructs and returns the corresponding header file name.
3. If the file_name doesn't match either format, it throws an error.

Value

The correct header file name as a `character` string.

A character string representing the header file name

See Also

`data_file`, `strip_extension` for related file name manipulation

Examples

```
fmt <- new("FileFormat", header_extension = "hdr", data_extension = "img")
header_file(fmt, "brain_scan.hdr")  # Returns "brain_scan.hdr"
header_file(fmt, "brain_scan.img")  # Returns "brain_scan.hdr"
```

`header_file_matches` *Generic function to test whether a file name conforms to the given `FileFormat` instance. Will test for match to header file only*

Description

Validates whether a file name conforms to the header file format specification.

Usage

```
header_file_matches(x, file_name)

## S4 method for signature 'FileFormat,character'
header_file_matches(x, file_name)
```

Arguments

`x`	A FileFormat object specifying the format requirements
`file_name`	A character string specifying the file name to validate

Details

The function performs case-sensitive pattern matching to verify that the file name ends with the specified header extension. The match is performed using a regular expression that ensures the extension appears at the end of the file name.

Value

`TRUE` for match, `FALSE` otherwise.

A logical value: `TRUE` if the file name matches the header format, `FALSE` otherwise

See Also

`file_matches`, `data_file_matches` for related file format validation

Examples

```
fmt <- new("FileFormat", header_extension = "hdr", data_extension = "img")
header_file_matches(fmt, "brain_scan.hdr")  # TRUE
header_file_matches(fmt, "brain_scan.img")  # FALSE
header_file_matches(fmt, "brain.hdr.gz")    # FALSE
```

`image` *Generic Image Method for Creating Visual Representations*

Description

Creates a visual representation (or image) from an object.

Arguments

`x`	An object to be rendered as an image.
`...`	Additional arguments passed to methods.

Value

An image object representing `x`.

`IndexLookupVol-class` *IndexLookupVol Class*

Description

A three-dimensional brain image class that serves as a map between 1D grid indices and a table of values. This class is primarily used in conjunction with the `SparseNeuroVec` class to efficiently represent and access sparse neuroimaging data.

The `IndexLookupVol` class provides efficient indexing and coordinate lookup functionality for 3D neuroimaging data. It maintains a mapping between linear indices and 3D coordinates, optimizing memory usage and access speed for sparse volumes.

Creates an `IndexLookupVol` object, which provides efficient bidirectional mapping between linear indices and 3D coordinates in a neuroimaging volume. This is particularly useful for working with masked or sparse brain volumes.

Usage

```
IndexLookupVol(space, indices)
```

Arguments

`space`	A `NeuroSpace` object defining the 3D space dimensions, spacing, and orientation.
`indices`	An integer vector containing the linear indices of the voxels to include in the lookup volume. These should be 1-based indices within the range of the space.

Details

The IndexLookupVol class extends `NeuroVol` and provides a mechanism for efficient lookup and mapping of sparse 3D neuroimaging data. It stores only the indices of non-zero voxels and their corresponding mappings, allowing for memory-efficient representation of large, sparse brain images.

Create an IndexLookupVol Object

Value

An object of class `IndexLookupVol` containing:

- A mapping between linear indices and sparse positions
- The original space information
- The subset of included voxel indices

Slots

`space` A `NeuroSpace` object representing the 3D space of the brain image.

`indices` An integer vector containing the 1D indices of the non-zero voxels in the grid.

`map` An integer vector containing the mapping between the 1D indices and the table of values.

Methods

This class inherits methods from `NeuroVol`. Additional methods specific to index lookup and mapping operations may be available.

Implementation Details

The class uses an integer mapping array for O(1) lookups between linear indices and their corresponding positions in the sparse representation.

See Also

`SparseNeuroVec-class` for the primary class that utilizes IndexLookupVol. `NeuroVol-class` for the base volumetric image class.

`IndexLookupVol` for creating instances of this class

`coords` for coordinate lookup, `lookup` for index mapping, `NeuroSpace` for space representation

Examples

```
# Create a NeuroSpace object
space <- NeuroSpace(dim = c(2L, 2L, 2L), origin = c(0, 0, 0), spacing = c(1, 1, 1))

# Create a 3D mask
mask <- array(c(TRUE, FALSE, TRUE, FALSE, FALSE, TRUE, FALSE, TRUE), dim = c(2, 2, 2))

# Create indices and map for the IndexLookupVol
indices <- which(mask)
map <- seq_along(indices)

# Create an IndexLookupVol object
ilv <- IndexLookupVol(space = space, indices = as.integer(indices))

# Access the indices
print(ilv@indices)

# Access the map
print(ilv@map)


# Create a 64x64x64 space
space <- NeuroSpace(c(64, 64, 64), c(1, 1, 1), c(0, 0, 0))

# Create a lookup volume with random indices
indices <- sample(1:262144, 10000)  # Select 10000 random voxels
ilv <- IndexLookupVol(space, indices)

# Look up coordinates for specific indices
coords <- coords(ilv, indices[1:10])
```

`index_to_coord` *convert 1d indices to n-dimensional real world coordinates*

Description

convert 1d indices to n-dimensional real world coordinates

Usage

```
index_to_coord(x, idx)

## S4 method for signature 'NeuroSpace,numeric'
index_to_coord(x, idx)

## S4 method for signature 'NeuroSpace,integer'
index_to_coord(x, idx)
```

```
## S4 method for signature 'NeuroVol,integer'
index_to_coord(x, idx)

## S4 method for signature 'NeuroVec,integer'
index_to_coord(x, idx)
```

Arguments

`x`	the object
`idx`	the 1D indices

Value

A numeric `matrix` of real-world coordinates.

Examples

```
bvol <- NeuroVol(array(0, c(10,10,10)), NeuroSpace(c(10,10,10), c(1,1,1)))
idx <- 1:10
g <- index_to_coord(bvol, idx)
idx2 <- coord_to_index(bvol, g)
all.equal(idx, idx2)
```

`index_to_grid` *Convert 1d indices to n-dimensional grid coordinates*

Description

Converts linear indices to 2D grid coordinates for a `NeuroSlice` object.

Usage

```
index_to_grid(x, idx)

## S4 method for signature 'NeuroSlice,numeric'
index_to_grid(x, idx)

## S4 method for signature 'NeuroSpace,numeric'
index_to_grid(x, idx)

## S4 method for signature 'NeuroVec,index'
index_to_grid(x, idx)

## S4 method for signature 'NeuroVec,integer'
index_to_grid(x, idx)

## S4 method for signature 'NeuroVol,index'
```

```
index_to_grid(x, idx)

## S4 method for signature 'NeuroVol,integer'
index_to_grid(x, idx)
```

Arguments

`x`	A `NeuroSlice` object
`idx`	Integer vector of linear indices to convert

Details

Convert Linear Indices to Grid Coordinates

Value

A numeric `matrix` of grid coordinates.

See Also

`grid_to_index` for the inverse operation

Examples

```
 bvol <- NeuroVol(array(0, c(10,10,10)), NeuroSpace(c(10,10,10), c(1,1,1)))
 idx <- 1:10
 g <- index_to_grid(bvol, idx)
 bvol[g]

slice_space <- NeuroSpace(c(10, 10))
slice_data <- matrix(1:100, 10, 10)
slice <- NeuroSlice(slice_data, slice_space)

# Convert single index
coords <- index_to_grid(slice, 55)

# Convert multiple indices
indices <- c(1, 25, 50, 75, 100)
coords_mat <- index_to_grid(slice, indices)
```

`indices` *Extract indices*

Description

Extract indices

Usage

```
indices(x)
```

Arguments

`x`	the object to extract indices

Value

A `vector` of indices from `x`.

Examples

```
# Create a NeuroSpace object with 3mm voxels
space <- NeuroSpace(c(10,10,10), spacing=c(3,3,3))

# Create ROI coordinates in voxel space
coords <- matrix(c(1,1,1, 2,2,2), ncol=3, byrow=TRUE)

# Create ROI volume
roi_vol <- ROIVol(space, coords, data=c(1,2))

# Get linear indices of ROI voxels
idx <- indices(roi_vol)
# These indices can be used to index into a 3D array of size 10x10x10
```

`indices,IndexLookupVol-method`

Get Indices from an IndexLookupVol Object

Description

Retrieves the vector of indices that are included in the lookup volume.

Usage

```
## S4 method for signature 'IndexLookupVol'
indices(x)

## S4 method for signature 'ROIVol'
indices(x)

## S4 method for signature 'ROIVol'
indices(x)

## S4 method for signature 'ROIVec'
indices(x)
```

```
## S4 method for signature 'AbstractSparseNeuroVec'
indices(x)
```

Arguments

x	An `IndexLookupVol` object

Value

the indices of the lookup volume

Examples

```
space <- NeuroSpace(c(64, 64, 64), c(1, 1, 1), c(0, 0, 0))
ilv <- IndexLookupVol(space, c(1:100))
idx <- indices(ilv)  # Get included indices
```

`inverse_trans` *Extract inverse image coordinate transformation*

Description

Extract inverse image coordinate transformation

Usage

```
inverse_trans(x)

## S4 method for signature 'NeuroSpace'
inverse_trans(x)
```

Arguments

x	an object

Value

A numeric 4x4 `matrix` that maps from real-world coordinates back to grid coordinates.

Examples

```
bspace <- NeuroSpace(c(10,10,10), c(2,2,2))
itrans <- inverse_trans(bspace)
identical(trans(bspace) %*% inverse_trans(bspace), diag(4))
```

`Kernel` *Create a Kernel object from a function of distance from kernel center*

Description

This function creates a Kernel object using a kernel function (`FUN`) that takes the distance from the center of the kernel as its first argument.

Usage

```
Kernel(kerndim, vdim, FUN = dnorm, ...)
```

Arguments

`kerndim`	A numeric vector representing the dimensions in voxels of the kernel.
`vdim`	A numeric vector representing the dimensions of the voxels in real units.
`FUN`	The kernel function taking its first argument representing the distance from the center of the kernel (default: `dnorm`).
`...`	Additional parameters to the kernel function, `FUN`.

Value

A Kernel object with the specified dimensions, voxel dimensions, and kernel function.

Examples

```
kdim <- c(3, 3, 3)
vdim <- c(1, 1, 1)
k <- Kernel(kerndim = kdim, vdim = vdim, FUN = dnorm, sd = 1)
```

`Kernel-class` *Kernel*

Description

A class representing an image kernel for image processing, such as convolution or filtering operations in brain images.

Slots

- `width` A `numeric` value representing the width of the kernel in voxels. The width is typically an odd number to maintain symmetry.
- `weights` A `numeric` vector containing the weights associated with each voxel in the kernel.
- `voxels` A `matrix` containing the relative voxel coordinates of the kernel. Each row represents a voxel coordinate as (x, y, z).
- `coords` A `matrix` containing the relative real-world coordinates of the kernel, corresponding to the voxel coordinates.

`labels,ClusteredNeuroVec-method`
Get Labels from ClusteredNeuroVec

Description

Get Labels from ClusteredNeuroVec

Usage

```
## S4 method for signature 'ClusteredNeuroVec'
labels(object)
```

Arguments

`object`	A ClusteredNeuroVec object

`laplace_enhance` *Laplacian Enhancement Filter for Volumetric Images*

Description

This function applies a multi-layer Laplacian enhancement filter to a volumetric image (3D brain MRI data). The filter enhances details while preserving edges using a non-local means approach with multiple scales.

Usage

```
laplace_enhance(
  vol,
  mask,
  k = 2,
  patch_size = 3,
  search_radius = 2,
  h = 0.7,
  mapping_params = NULL,
  use_normalization_free = TRUE
)
```

Arguments

`vol`	A `NeuroVol` object representing the image volume to be enhanced.
`mask`	A `LogicalNeuroVol` object specifying the region to process. If not provided, the entire volume will be processed.
`k`	An integer specifying the number of layers in the decomposition (default is 2).

`patch_size`	An integer specifying the size of patches for non-local means. Must be odd (default is 3).
`search_radius`	An integer specifying the radius of the search window (default is 2).
`h`	A numeric value controlling the filtering strength. Higher values mean more smoothing (default is 0.7).
`mapping_params`	An optional list of parameters for the enhancement mappings.
`use_normalization_free`	Logical indicating whether to use normalization-free weights (default is TRUE).

Value

A `NeuroVol` object representing the enhanced image.

`length,ClusteredNeuroVec-method`

Get length of NeuroVec object

Description

Returns the number of time points (4th dimension) in a NeuroVec object. This represents the temporal dimension of the neuroimaging data.

Returns the total number of time points across all vectors in the sequence

Usage

```
## S4 method for signature 'ClusteredNeuroVec'
length(x)

## S4 method for signature 'NeuroVec'
length(x)

## S4 method for signature 'NeuroVecSeq'
length(x)

## S4 method for signature 'ROIVol'
length(x)

## S4 method for signature 'ROICoords'
length(x)
```

Arguments

`x`	A NeuroVecSeq object

Value

Integer length (total number of time points)

An integer representing the number of coordinates in the ROICoords object.

`linear_access` *Extract values from an array-like object using linear indexing.*

Description

This function extracts the values of the elements in an array-like object using linear indexing. Linear indexing is a way of indexing an array by a single index that is computed from multiple indices using a formula.

Usage

```
linear_access(x, i, ...)
```

Arguments

`x`	a data source.
`i`	a vector of indices.
`...`	additional arguments to be passed to methods.

Value

A `vector` containing the values at the specified linear indices of `x`.

Examples

```
# Create a sparse neuroimaging vector
bspace <- NeuroSpace(c(10,10,10,100), c(1,1,1))
mask <- array(rnorm(10*10*10) > .5, c(10,10,10))
mat <- matrix(rnorm(sum(mask)), 100, sum(mask))
svec <- SparseNeuroVec(mat, bspace, mask)

# Extract values using linear indices
# Get values from first timepoint at voxels 1,2,3
indices <- c(1,2,3)
vals <- linear_access(svec, indices)

# Get values from multiple timepoints and voxels
# First voxel at timepoint 1, second voxel at timepoint 2
indices <- c(1, 1000 + 2) # 1000 = prod(10,10,10)
vals <- linear_access(svec, indices)
```

`linear_access,DenseNeuroVol,numeric-method`

Linear Access Method for FileBackedNeuroVec

Description

Internal method providing linear access to memory-mapped data.

Provides linear access to the data across all vectors in the sequence.

Usage

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

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

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

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

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

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

## S4 method for signature 'NeuroHyperVec,ANY'
linear_access(x, i, ...)

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

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

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

Arguments

`x`	A NeuroVecSeq object
`i`	Numeric vector of indices for linear access
`...`	Additional arguments (not used)

Value

Numeric vector of accessed values

Examples

```
# Create a small NeuroVec and save it
nvec <- NeuroVec(matrix(1:32, 8, 4), NeuroSpace(c(2,2,2,4)))
tmp <- tempfile(fileext = ".nii")
write_vec(nvec, tmp)

# Load as FileBackedNeuroVec and access values
fbvec <- FileBackedNeuroVec(tmp)
values <- linear_access(fbvec, 1:10)

# Clean up
unlink(tmp)
```

`list_afni_attributes` *List AFNI Attributes in Extension*

Description

Returns a character vector of all attribute names in an AFNI extension.

Usage

```
list_afni_attributes(ext)
```

Arguments

`ext`	A `NiftiExtension-class` object with ecode = 4, or an xml_document from `parse_afni_extension`.

Value

Character vector of attribute names.

Examples

```
## Not run:
# List all attributes in an AFNI extension
attrs <- list_afni_attributes(afni_ext)
print(attrs)

## End(Not run)
```

`load_data,MappedNeuroVecSource-method`

Load image data from a NeuroVecSource object

Description

This function loads the image data from a NeuroVecSource object, handling various dimensionalities and applying any necessary transformations.

Usage

```
## S4 method for signature 'MappedNeuroVecSource'
load_data(x)

## S4 method for signature 'NeuroVecSource'
load_data(x)

## S4 method for signature 'NeuroVolSource'
load_data(x)

## S4 method for signature 'SparseNeuroVecSource'
load_data(x)
```

Arguments

`x`	The NeuroVecSource object containing the image metadata and file information.

Details

This method performs the following steps: 1. Validates the dimensionality of the metadata. 2. Reads the image data using RNifti. 3. Handles 5D arrays by dropping the 4th dimension if it has length 1. 4. Applies slope scaling if present in the metadata. 5. Constructs a NeuroSpace object with appropriate dimensions and spatial information. 6. Creates and returns a DenseNeuroVec object, handling both 3D and 4D input arrays.

Value

a DenseNeuroVec object

Note

This method currently only supports NIfTI file format through RNifti.

See Also

`NeuroVecSource`, `DenseNeuroVec`, `NeuroSpace`

`LogicalNeuroVol-class` *LogicalNeuroVol Class*

Description

This class represents a three-dimensional brain image where all values are either TRUE or FALSE. It is particularly useful for creating and managing binary masks for brain images.

This function constructs a `LogicalNeuroVol` instance.

Usage

```
LogicalNeuroVol(data, space, label = "", indices = NULL)
```

Arguments

`data`	A three-dimensional `array`, a 1D vector with length equal to `prod(dim(space))`, or a set of `indices` where elements are `TRUE`.
`space`	An instance of class `NeuroSpace`.
`label`	A `character` string.
`indices`	An optional 1-d index vector.

Details

The LogicalNeuroVol class extends the `DenseNeuroVol` class, inheriting its spatial properties and array-based storage. However, it constrains the values to be logical (TRUE or FALSE), making it ideal for representing binary masks, regions of interest (ROIs), or segmentation results in neuroimaging analyses.

Value

A `LogicalNeuroVol` instance.

Slots

`.Data` A logical array containing the binary volume data.

`space` A `NeuroSpace` object defining the spatial properties of the volume.

Methods

This class inherits methods from `DenseNeuroVol`. Additional methods specific to logical operations may be available.

See Also

`DenseNeuroVol-class` for the parent class. `NeuroVol-class` for the base volumetric image class.

Examples

```
# Create a simple logical brain volume (e.g., a mask)
dim <- c(64L, 64L, 64L)
mask_data <- array(sample(c(TRUE, FALSE), prod(dim), replace = TRUE), dim)
mask_space <- NeuroSpace(dim = dim, origin = c(0, 0, 0), spacing = c(1, 1, 1))
brain_mask <- new("LogicalNeuroVol", .Data = mask_data, space = mask_space)

# Check the proportion of TRUE voxels
true_proportion <- sum(brain_mask) / prod(dim(brain_mask))
print(paste("Proportion of TRUE voxels:", true_proportion))

# Load an example brain mask
brain_mask <- read_vol(system.file("extdata", "global_mask_v4.nii", package="neuroim2"))

# Convert the brain mask to a LogicalNeuroVol
logical_vol <- LogicalNeuroVol(brain_mask, space(brain_mask))
```

`lookup` *Index Lookup operation*

Description

Index Lookup operation

Usage

```
lookup(x, i, ...)
```

Arguments

`x`	the object to query
`i`	the index to lookup
`...`	additional arguments

Value

The value(s) at the specified index/indices of `x`.

Examples

```
# Create a 64x64x64 space
space <- NeuroSpace(c(64, 64, 64), c(1, 1, 1), c(0, 0, 0))

# Create a lookup volume with first 100 indices
ilv <- IndexLookupVol(space, 1:100)

# Look up values for indices 1, 2, and 3
```

```
# Returns their positions in the sparse representation
lookup(ilv, c(1, 2, 3))

# Look up values outside the included indices
# Returns 0 for indices not in the lookup volume
lookup(ilv, c(101, 102))
```

`lookup,IndexLookupVol,numeric-method`

Lookup Values in an IndexLookupVol Object

Description

Performs a lookup operation on an IndexLookupVol object.

Usage

```
## S4 method for signature 'IndexLookupVol,numeric'
lookup(x, i)

## S4 method for signature 'AbstractSparseNeuroVec,numeric'
lookup(x, i)
```

Arguments

`x`	An `IndexLookupVol` object
`i`	A numeric vector of indices to look up

Value

the values of the lookup volume

Examples

```
space <- NeuroSpace(c(64, 64, 64), c(1, 1, 1), c(0, 0, 0))
ilv <- IndexLookupVol(space, c(1:100))
lookup(ilv, c(1, 2, 3))  # Look up values for indices 1, 2, and 3
```

`make_time_weights` *Build smooth time weights from motion/outlier metrics*

Description

Creates per-time-point weights $w_t \in [0, 1]$ by smoothly combining framewise displacement (FD), DVARS, and spike/outlier scores. Each series is transformed through a soft logistic ramp so that values beyond the specified thresholds receive progressively lower weights instead of hard 0/1 decisions.

Usage

```
make_time_weights(
  fd = NULL,
  dvars = NULL,
  spike = NULL,
  fd_thr = 0.5,
  dvars_z = 2.5,
  spike_z = 5,
  fd_soft = 0.1,
  dvars_soft = 0.25,
  combine = c("min", "prod")
)
```

Arguments

`fd`	Optional numeric vector of framewise displacement values.
`dvars`	Optional numeric vector of DVARS values.
`spike`	Optional numeric vector with spike/outlier magnitudes.
`fd_thr`	Threshold (in mm) where FD weights start to drop (default 0.5).
`dvars_z`	Z-threshold applied to the standardized DVARS series (default 2.5).
`spike_z`	Z-threshold applied to the standardized spike series (default 5).
`fd_soft`	Logistic softness (in mm) controlling the slope around `fd_thr`.
`dvars_soft`	Logistic softness for the DVARS z-scores.
`combine`	Either `"min"` (take the minimum weight per TR) or `"prod"` (multiply all weights).

Value

Numeric vector of weights in $[0, 1]$ with length equal to the provided series. At least one of `fd`, `dvars`, or `spike` must be supplied.

`mapf` *Apply a function to an object.*

Description

This function applies a function to an object, with additional arguments passed to the function using the `...` argument. The mapping object specifies how the function is to be applied, and can take many different forms, depending on the object and function used. The return value depends on the function used.

Usage

```
mapf(x, m, ...)

## S4 method for signature 'NeuroVol,Kernel'
mapf(x, m, mask = NULL)
```

Arguments

`x`	the object that is mapped.
`m`	the mapping object.
`...`	additional arguments to be passed to the function.
`mask`	restrict application of kernel to masked area

Value

The result of applying the mapping function to `x`.

Examples

```
# Create a simple 3D volume
bspace <- NeuroSpace(c(10,10,10), c(1,1,1))
vol <- NeuroVol(array(rnorm(10*10*10), c(10,10,10)), bspace)

# Create a 3x3x3 mean smoothing kernel
kern <- Kernel(c(3,3,3),  vdim=c(3,3,3))

# Apply the kernel to smooth the volume
smoothed_vol <- mapf(vol, kern)
```

`MappedNeuroVec-class` *MappedNeuroVec Class*

Description

A class representing a four-dimensional brain image backed by a memory-mapped file. This class provides efficient access to large brain images without loading the entire dataset into memory.

The `MappedNeuroVec` class provides memory-efficient access to large neuroimaging datasets through memory mapping. This allows processing of datasets larger than available RAM by keeping data on disk and only loading requested portions into memory.

Creates a `MappedNeuroVec` object that provides efficient, memory-mapped access to large neuroimaging datasets. This allows processing of data larger than available RAM by keeping it on disk and only loading requested portions into memory.

Usage

```
MappedNeuroVec(file_name, label = basename(file_name))
```

Arguments

`file_name`	Character string specifying the path to the neuroimaging file. Supported formats include NIFTI (.nii) and ANALYZE (.hdr/.img).
`label`	Optional character string providing a label for the vector

Details

MappedNeuroVec objects use memory-mapped files to store and access large 4D brain images efficiently. This approach allows for rapid access to specific portions of the data without requiring the entire dataset to be loaded into memory at once.

Create a Memory-Mapped Neuroimaging Vector

The function implements several key features:

- Zero-copy access to file data
- Automatic memory management
- Support for large datasets
- Efficient random access
- Proper cleanup on object deletion

Memory mapping is particularly useful when:

- Working with large datasets
- Only portions of data are needed at once
- Random access is required
- Multiple processes need to share data

Value

A new `MappedNeuroVec` object providing:

- Memory-mapped access to the data
- Spatial and temporal indexing
- Efficient data extraction
- Automatic memory management

Slots

`filemap` An object of class `mmap` representing the memory-mapped file containing the brain image data.

`offset` An integer representing the byte offset within the memory-mapped file where the brain image data starts.

Methods

This class inherits methods from `NeuroVec` and implements the `ArrayLike4D` interface. Additional methods specific to memory-mapped operations may be available.

Implementation Details

The class uses the `mmap` package to establish a memory mapping between the file and memory space. Key features include:

- Zero-copy access to file data
- Automatic memory management
- Support for large datasets
- Efficient random access

See Also

`NeuroVec-class` for the parent class. `mmap` for details on memory-mapped file objects.

`MappedNeuroVec` for creating instances of this class

`mmap` for memory mapping details

Examples

```
# Create a MappedNeuroVec object (pseudo-code)
file_path <- system.file("extdata", "global_mask_v4.nii", package = "neuroim2")
mapped_vec <- MappedNeuroVec(file_path)

# Access a subset of the data
subset <- mapped_vec[,,, 1:2]


# Create mapped vector from NIFTI file
mvec <- MappedNeuroVec(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))
```

```
# Extract first volume
vol1 <- mvec[[1]]

# Get dimensions
dim(mvec)

# Access specific timepoint
timepoint <- mvec[, , , 2]
```

MappedNeuroVecSource-class

MappedNeuroVecSource Class

Description

A class used to produce a `MappedNeuroVec` instance. It encapsulates the necessary information to create a memory-mapped representation of a 4D neuroimaging dataset.

Creates a `MappedNeuroVecSource` object that manages the memory mapping between a neuroimaging file and memory space. This is typically used internally by `MappedNeuroVec` but can be created directly for custom access patterns.

Usage

```
MappedNeuroVecSource(file_name)
```

Arguments

`file_name`	Character string specifying the path to the neuroimaging file. Supported formats include NIFTI (.nii) and ANALYZE (.hdr/.img).

Details

MappedNeuroVecSource acts as a factory for MappedNeuroVec objects. While it doesn't have any additional slots beyond its parent class, it specifies the intent to create a memory-mapped representation of the neuroimaging data. This class is typically used in data loading pipelines where large datasets need to be accessed efficiently without loading the entire dataset into memory.

Create a Memory-Mapped Source for Neuroimaging Data

The function performs several important checks:

- Validates file existence and permissions
- Reads and validates header information
- Ensures proper dimensionality (>= 3D)
- Verifies file format compatibility

Value

A new `MappedNeuroVecSource` object containing:

- Meta information about the dataset
- File format details
- Dimensional information

Inheritance

`MappedNeuroVecSource` inherits from:

- `NeuroVecSource`: Base class for NeuroVec source objects

See Also

`MappedNeuroVec` for the main user interface, `read_header` for header reading details

Examples

```
# Create a MappedNeuroVecSource
mapped_source <- new("MappedNeuroVecSource")


# Create source from NIFTI file
source <- MappedNeuroVecSource(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))

# Check dimensions
dim(source@meta_info)

# View header information
str(source@meta_info)
```

`mapToColors` *Map intensity values to colors*

Description

Convert intensity values (e.g., a 2D slice) into a color representation for plotting and overlays.

Usage

```
mapToColors(
  imslice,
  col = heat.colors(128, alpha = 1),
  zero_col = "#00000000",
  alpha = 1,
```

```
  irange = range(imslice),
  threshold = c(0, 0)
)
```

Arguments

`imslice`	A numeric vector or array of intensities.
`col`	A vector of colors used as a lookup table.
`zero_col`	Color used for exactly-zero intensities (defaults to transparent).
`alpha`	Global alpha multiplier applied to all colors when `alpha` < `1`.
`irange`	Intensity range used to normalize values before mapping to `col`.
`threshold`	Optional length-2 numeric vector. If `diff(threshold)` > `0`, values within `[threshold[1], threshold[2]]` are set to transparent.

Value

If `alpha` == `1`, returns a character vector/array of colors. If `alpha` < `1`, returns an array with an added RGBA channel (last dimension length 4).

`map_values` *Map Values from One Set to Another Using a User-supplied Lookup Table*

Description

This function maps values from one set to another using a lookup table provided by the user.

Usage

```
map_values(x, lookup)

## S4 method for signature 'NeuroVol,list'
map_values(x, lookup)

## S4 method for signature 'NeuroVol,matrix'
map_values(x, lookup)
```

Arguments

`x`	The object from which values will be mapped.
`lookup`	The lookup table. The first column is the "key" and the second column is the "value".

Value

An object of the same class as `x`, in which the original values have been replaced with the lookup table values.

Examples

```
x <- NeuroSpace(c(10, 10, 10), c(1, 1, 1))
vol <- NeuroVol(sample(1:10, 10 * 10 * 10, replace = TRUE), x)

## Lookup table is a list
lookup <- lapply(1:10, function(i) i * 10)
names(lookup) <- 1:10
ovol <- map_values(vol, lookup)

## Lookup table is a matrix. The first column is the key, and the second column is the value
names(lookup) <- 1:length(lookup)
lookup.mat <- cbind(as.numeric(names(lookup)), unlist(lookup))
ovol2 <- map_values(vol, lookup.mat)
all.equal(as.vector(ovol2), as.vector(ovol))
```

`mask` *Extract Mask from Neuroimaging Object*

Description

Generic function to extract or generate a mask from neuroimaging objects. For sparse objects with a `@mask` slot, returns the stored mask. For dense objects, returns a filled mask (all TRUE values) indicating all voxels contain valid data.

Usage

```
mask(x)

## S4 method for signature 'ClusteredNeuroVol'
mask(x)

## S4 method for signature 'FileBackedNeuroVec'
mask(x)

## S4 method for signature 'MappedNeuroVec'
mask(x)

## S4 method for signature 'NeuroHyperVec'
mask(x)

## S4 method for signature 'NeuroSlice'
mask(x)

## S4 method for signature 'DenseNeuroVec'
mask(x)

## S4 method for signature 'DenseNeuroVol'
```

```
mask(x)

## S4 method for signature 'LogicalNeuroVol'
mask(x)

## S4 method for signature 'AbstractSparseNeuroVec'
mask(x)

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

Arguments

`x`	A neuroimaging object (NeuroVol, NeuroVec, or derived classes)

Details

The behavior depends on the class of the input object:

- For sparse objects (SparseNeuroVec, ClusteredNeuroVol, etc.): Returns the stored `@mask` slot
- For dense objects (DenseNeuroVol, DenseNeuroVec, etc.): Returns a LogicalNeuroVol with all TRUE values
- For ROI objects: Not implemented (use `coords()` instead)

Value

A `LogicalNeuroVol` object representing the mask

Examples

```
# Create a dense volume
vol <- NeuroVol(array(rnorm(64^3), c(64,64,64)), NeuroSpace(c(64,64,64)))
m <- mask(vol)  # Returns all TRUE mask

# Create a sparse vector with explicit mask
mask_array <- array(runif(64^3) > 0.5, c(64,64,64))
mask_vol <- LogicalNeuroVol(mask_array, NeuroSpace(c(64,64,64)))
# Data must be a matrix (time x masked voxels)
sparse_data <- matrix(rnorm(sum(mask_array) * 10), nrow = 10, ncol = sum(mask_array))
svec <- SparseNeuroVec(sparse_data, NeuroSpace(c(64,64,64,10)), mask_vol)
m2 <- mask(svec)  # Returns the stored mask
```

`matricized_access` *Extract values from a 4D tensor using a matrix of time-space indices.*

Description

This function efficiently extracts values from a 4D tensor (typically neuroimaging data) using a matrix of indices where each row contains a time index in column 1 and a spatial index in column 2. The spatial index refers to the position in the flattened spatial dimensions (x,y,z). This is primarily used internally by the `series()` method to efficiently access time series data for specific voxels.

Usage

```
matricized_access(x, i, ...)

## S4 method for signature 'SparseNeuroVec,matrix'
matricized_access(x, i)

## S4 method for signature 'SparseNeuroVec,integer'
matricized_access(x, i)

## S4 method for signature 'SparseNeuroVec,numeric'
matricized_access(x, i)

## S4 method for signature 'BigNeuroVec,matrix'
matricized_access(x, i)

## S4 method for signature 'BigNeuroVec,integer'
matricized_access(x, i)

## S4 method for signature 'BigNeuroVec,numeric'
matricized_access(x, i)
```

Arguments

`x`	a data source, typically a `SparseNeuroVec` object containing 4D neuroimaging data
`i`	Either: • A matrix with 2 columns: [time_index, space_index] specifying which values to extract • A numeric vector of spatial indices to extract all timepoints for those locations
`...`	additional arguments to be passed to methods.

Value

When `i` is a matrix, returns a numeric vector of values at the specified time-space coordinates. When `i` is a vector, returns a matrix where each column contains the full time series for each spatial index.

Examples

```
# Create a sparse 4D neuroimaging vector
bspace <- NeuroSpace(c(10,10,10,100), c(1,1,1))
mask <- array(rnorm(10*10*10) > .5, c(10,10,10))
mat <- matrix(rnorm(sum(mask)), 100, sum(mask))
svec <- SparseNeuroVec(mat, bspace, mask)

# Extract specific timepoint-voxel pairs
# Get value at timepoint 1, voxel 1 and timepoint 2, voxel 2
idx_mat <- matrix(c(1,1, 2,2), ncol=2, byrow=TRUE)
vals <- matricized_access(svec, idx_mat)

# Get full time series for voxels 1 and 2
ts_mat <- matricized_access(svec, c(1,2))
# Each column in ts_mat contains the full time series for that voxel
```

`matrixToQuatern` *Convert a Transformation Matrix to a Quaternion Representation*

Description

Extracts the rotation and scaling components from a 3x3 (or 4x4) transformation matrix, normalizes them, and computes the corresponding quaternion parameters and a sign factor (‘qfac‘) indicating whether the determinant is negative.

Usage

```
matrixToQuatern(mat)
```

Arguments

`mat`	A numeric matrix with at least the top-left 3x3 portion containing rotation/scaling. Often a 4x4 affine transform, but only the 3x3 top-left submatrix is used in practice.

Details

This function first checks and corrects for zero-length axes in the upper-left corner of the matrix, then normalizes each column to extract the pure rotation. If the determinant of the rotation submatrix is negative, the `qfac` is set to `-1`, and the third column is negated. Finally, the quaternion parameters (a, b, c, d) are computed following standard NIfTI-1 conventions for representing the rotation in 3D.

Value

A named `list` with two elements:

`quaternion` A numeric vector of length 3, (b, c, d), which—together with a derived internally—represents the rotation.

`qfac` Either `+1` or `-1`, indicating whether the determinant of the rotation submatrix is positive or negative, respectively.

References

- Cox RW. *Analysis of Functional NeuroImages* (AFNI) and NIfTI-1 quaternion conventions. `https://afni.nimh.nih.gov`

See Also

`quaternToMatrix` for the inverse operation, converting quaternion parameters back to a transform matrix.

`MetaInfo` *Create Neuroimaging Metadata Object*

Description

Creates a MetaInfo object containing essential metadata for neuroimaging data, including dimensions, spacing, orientation, and data type information.

Usage

```
MetaInfo(
  Dim,
  spacing,
  origin = rep(0, length(spacing)),
  data_type = "FLOAT",
  label = "",
  spatial_axes = OrientationList3D$AXIAL_LPI,
  additional_axes = NullAxis
)
```

Arguments

`Dim`	Integer vector. Image dimensions (e.g., c(64, 64, 32) for 3D).
`spacing`	Numeric vector. Voxel dimensions in mm.
`origin`	Numeric vector. Coordinate origin. Default is zero vector.
`data_type`	Character. Data type (e.g., "FLOAT", "SHORT"). Default is "FLOAT".
`label`	Character. Image label(s). Default is "".
`spatial_axes`	Object. Spatial orientation. Default is OrientationList3D$AXIAL_LPI.
`additional_axes`	Object. Non-spatial axes. Default is NullAxis.

Details

Create MetaInfo Object

The MetaInfo object is fundamental for:

- Spatial interpretation of image data
- Data type handling and conversion
- Memory allocation and mapping
- File I/O operations

Input validation ensures:

- Dimensions are positive integers
- Spacing values are positive
- Origin coordinates are finite
- Data type is supported

Value

A MetaInfo object

See Also

`NIFTIMetaInfo`, `AFNIMetaInfo`

Examples

```
# Create metadata for 3D structural MRI
meta <- MetaInfo(
  Dim = c(256, 256, 180),
  spacing = c(1, 1, 1),
  data_type = "FLOAT",
  label = "T1w"
)

# Get image dimensions
dim(meta)

# Get transformation matrix
trans(meta)
```

`MetaInfo-class` *MetaInfo Class*

Description

This class encapsulates meta information for neuroimaging data types, including spatial and temporal characteristics, data type, and labeling.

Details

The MetaInfo class provides a structured way to store and access essential metadata for neuroimaging data. This includes information about the data type, spatial and temporal dimensions, voxel spacing, and coordinate system origin.

Slots

- `data_type` A `character` string specifying the data type code (e.g., "FLOAT", "INT").
- `dims` A `numeric` vector representing image dimensions.
- `spatial_axes` An `AxisSet3D` object representing image axes for spatial dimensions (x, y, z).
- `additional_axes` An `AxisSet` object representing axes for dimensions beyond spatial (e.g., time, color band, direction).
- `spacing` A `numeric` vector representing voxel dimensions in real-world units.
- `origin` A `numeric` vector representing the coordinate origin.
- `label` A `character` vector containing name(s) of images or data series.

See Also

`FileMetaInfo-class`, `AxisSet3D-class`, `AxisSet-class`

Examples

```
# Create a MetaInfo object
meta_info <- new("MetaInfo",
                 data_type = "FLOAT",
                 dims = c(64, 64, 32, 100),
                 spatial_axes = new("AxisSet3D"),
                 additional_axes = new("AxisSet"),
                 spacing = c(3, 3, 4),
                 origin = c(0, 0, 0),
                 label = "fMRI_run1")
```

`meta_info` *Lightweight metadata for neuroimaging files*

Description

‘meta_info()‘ provides a simple, CRAN-friendly way to retrieve essential image metadata without teaching S4 details up front. It accepts a file path or a ‘FileMetaInfo‘ object and returns a normalized list containing common fields like dimensions, spacing, origin, and transform.

The function does not read image data; it only parses header information.

Usage

```
meta_info(x)

## S4 method for signature 'FileMetaInfo'
meta_info(x)

## S4 method for signature 'character'
meta_info(x)
```

Arguments

`x`	A character file path (e.g., ‘"image.nii.gz"‘) or an object of class `FileMetaInfo`.

Details

Summarize Image Metadata

Value

A named list with the following elements:

- ‘dim‘ Integer vector of image dimensions.
- ‘spacing‘ Numeric voxel spacing (mm).
- ‘origin‘ Numeric coordinate origin.
- ‘trans‘ 4x4 transformation matrix mapping grid to world (mm).
- ‘path‘ Data file path.
- ‘filename‘ Basename of ‘path‘.
- ‘format‘ File format label (e.g., "NIFTI", "AFNI").
- ‘dtype‘ Storage data type label.
- ‘bytes_per_element‘ Bytes per element.
- ‘nvox‘ Number of voxels in the spatial volume (prod of first 3 dims).
- ‘nvol‘ Number of volumes (4th dim if present, else 1).
- ‘size_bytes‘ Approximate uncompressed size in bytes (‘nvox * nvol * bytes_per_element‘).
- ‘time_step‘ Time step (TR in seconds) if available for NIfTI, else ‘NA_real_‘.

See Also

`read_header`, `trans`, `FileMetaInfo`, `NIFTIMetaInfo`

Examples

```
f <- system.file("extdata", "global_mask_v4.nii", package = "neuroim2")
mi <- meta_info(f)
mi$dim
mi$spacing
mi$origin
mi$filename
# 4x4 transform
mi$trans
```

`NamedAxis-class` *NamedAxis*

Description

This class represents an axis with a name attribute

Slots

`axis` the name of the axis

`direction` of axis (-1,+1)

`ndim` *Extract the number of dimensions of an object*

Description

Extract the number of dimensions of an object

Usage

```
ndim(x, ...)

## S4 method for signature 'ClusteredNeuroVec'
ndim(x)

## S4 method for signature 'NeuroObj'
ndim(x)

## S4 method for signature 'NeuroSpace'
ndim(x)
```

Arguments

`x`	n-dimensional object
`...`	additional arguments

Value

An integer representing the number of dimensions in `x`.

Examples

```
x = NeuroSpace(c(10,10,10), spacing=c(1,1,1))
ndim(x) == 3
x = NeuroSpace(c(10,10,10,3), spacing=c(1,1,1))
ndim(x) == 4
```

`ndim,AxisSet-method` *Get number of dimensions in axis set*

Description

Get number of dimensions in axis set

Usage

```
## S4 method for signature 'AxisSet'
ndim(x, ...)
```

Arguments

`x`	An AxisSet object
`...`	Additional arguments (not used)

Value

An integer representing the number of dimensions in `x`.

`neuro-downsample` *Downsampling Methods for Neuroimaging Objects*

Description

Methods for downsampling neuroimaging objects to lower spatial resolution

---

`neuro-ops` *Arithmetic and Comparison Operations for Neuroimaging Objects*

Description

Methods for performing arithmetic and comparison operations on neuroimaging objects

---

`neuro-resample` *Resampling Methods for Neuroimaging Objects*

Description

Methods for resampling neuroimaging objects to different spaces and dimensions

---

`NeuroBucket-class` *NeuroBucket*

Description

a four-dimensional image that consists of a sequence of labeled image volumes backed by a list

Slots

`labels` the names of the sub-volumes contained in the bucket

`data` a list of `NeuroVol` instances with names corresponding to volume labels

---

`NeuroHyperVec` *Constructor for NeuroHyperVec class*

Description

Constructor for NeuroHyperVec class

Usage

```
NeuroHyperVec(data, space, mask)
```

Arguments

`data`	A matrix or three-dimensional array containing the data.
`space`	A `NeuroSpace` object defining the spatial dimensions.
`mask`	A mask volume (array, vector, or `LogicalNeuroVol`).

Value

A new NeuroHyperVec object.

See Also

NeuroSpace, LogicalNeuroVol

Examples

```
# Create a 5D space (10x10x10 spatial, 2 trials, 2 features)
space <- NeuroSpace(c(10,10,10,2,2))

# Create a mask for the spatial dimensions
space3d <- NeuroSpace(c(10,10,10))
mask_data <- array(TRUE, dim=c(10,10,10))  # All voxels active
mask <- LogicalNeuroVol(mask_data, space3d)

# Create data in the format expected by NeuroHyperVec:
# 3D array with dimensions [features x trials x voxels]
n_features <- 2
n_trials <- 2
n_voxels <- sum(mask_data)  # 1000 voxels
data_array <- array(rnorm(n_features * n_trials * n_voxels),
                   dim = c(n_features, n_trials, n_voxels))

# Create the NeuroHyperVec object
hvec <- NeuroHyperVec(data_array, space, mask)
```

`NeuroHyperVec-class` *NeuroHyperVec Class*

Description

A class representing a five-dimensional brain image, where the first three dimensions are spatial, the fourth dimension is typically time or trials, and the fifth dimension represents features within a trial.

The `NeuroHyperVec` class provides an efficient container for five-dimensional neuroimaging data where spatial dimensions are sparse. It is particularly suited for analyses involving multiple features per trial/timepoint, such as basis functions, spectral components, or multi-modal measurements.

Usage

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

Arguments

`x`	The NeuroHyperVec object
`i`, `j`, `k`, `l`, `m`	Indices for each dimension
`...`	Additional arguments (not used)
`drop`	Whether to drop dimensions of length 1

Details

Five-Dimensional Sparse Neuroimaging Data Container

The class organizes data in a 5D structure:

- Dimensions 1-3: Spatial coordinates (x, y, z)
- Dimension 4: Trials or timepoints
- Dimension 5: Features or measurements

Data is stored internally as a three-dimensional array for efficiency:

- Dimensions 1: Features (dimension 5)
- Dimensions 2: Trials (dimension 4)
- Dimensions 3: Voxels (flattened spatial)

Key features:

- Memory-efficient sparse storage of spatial dimensions
- Fast access to feature vectors and time series
- Flexible indexing across all dimensions
- Maintains spatial relationships and metadata

Slots

`mask` An object of class `LogicalNeuroVol` defining the sparse spatial domain of the brain image.

`data` A 3D array with dimensions [features x trials x voxels] containing the neuroimaging data.

`space` A `NeuroSpace` object representing the dimensions and voxel spacing of the neuroimaging data.

`lookup_map` An integer vector for O(1) spatial index lookups.

`mask` A `LogicalNeuroVol` object defining the spatial mask.

`data` A three-dimensional array with dimensions [features x trials x voxels] containing the data.

`space` A `NeuroSpace` object defining the 5D space.

`lookup_map` An integer vector for O(1) spatial index lookups.

See Also

`NeuroVec`, `LogicalNeuroVol`, `NeuroSpace`

Examples

```
# Create a simple 5D dataset (10x10x10 spatial, 5 trials, 3 features)
dims <- c(10, 10, 10)
space <- NeuroSpace(c(dims, 5, 3))

# Create a sparse mask (20% of voxels)
mask_data <- array(runif(prod(dims)) < 0.2, dims)
mask <- LogicalNeuroVol(mask_data, NeuroSpace(dims))

# Generate random data for active voxels
n_voxels <- sum(mask_data)
data <- array(rnorm(3 * 5 * n_voxels), dim = c(3, 5, n_voxels))  # [features x trials x voxels]

# Create NeuroHyperVec object
hvec <- NeuroHyperVec(data, space, mask)

# Access operations
# Get data for specific voxel across all trials/features
series(hvec, 5, 5, 5)

# Extract a 3D volume for specific trial and feature
hvec[,,,2,1]
```

`NeuroObj-class` *NeuroObj Class*

Description

Base class for all neuroimaging data objects with a Cartesian spatial representation. This class provides a foundation for more specific neuroimaging data structures.

Slots

`space` An object of class `NeuroSpace` representing the geometry of the image object.

See Also

`NeuroSpace-class`, `NeuroSlice-class`, `NeuroVol-class`

`NeuroSlice` *NeuroSlice: 2D Neuroimaging Data Container*

Description

Creates a `NeuroSlice` object representing a two-dimensional slice of neuroimaging data with associated spatial information. This class is particularly useful for working with individual slices from volumetric neuroimaging data or for visualizing 2D cross-sections.

Usage

```
NeuroSlice(data, space, indices = NULL)
```

Arguments

`data`	A vector or matrix containing the slice data values.
`space`	An object of class `NeuroSpace` defining the spatial properties (dimensions, spacing, origin) of the slice.
`indices`	Optional integer vector. When `data` is provided as a 1D vector, `indices` specifies the linear indices where the data values should be placed in the 2D slice. Useful for creating sparse slices. Default is NULL.

Details

Two-Dimensional Neuroimaging Data Slice

Value

A new object of class `NeuroSlice`.

Input Validation

The function performs several validation checks:

- Verifies that `space` is 2-dimensional
- Ensures data dimensions are compatible with `space`
- Validates `indices` when provided for sparse initialization

Data Handling

The function supports two initialization modes:

- Dense mode (indices = NULL):
  - Data is reshaped if necessary to match space dimensions
  - Dimensions must match exactly after reshaping
- Sparse mode (indices provided):
  - Creates a zero-initialized matrix matching space dimensions
  - Places data values at specified indices

See Also

NeuroSpace for defining spatial properties, NeuroVol for 3D volumetric data, plot for visualization methods

Examples

```
# Create a 64x64 slice space
slice_space <- NeuroSpace(c(64, 64), spacing = c(2, 2))

# Example 1: Dense slice from matrix
slice_data <- matrix(rnorm(64*64), 64, 64)
dense_slice <- NeuroSlice(slice_data, slice_space)

# Example 2: Dense slice from vector
vec_data <- rnorm(64*64)
vec_slice <- NeuroSlice(vec_data, slice_space)

# Example 3: Sparse slice with specific values
n_points <- 100
sparse_data <- rnorm(n_points)
sparse_indices <- sample(1:(64*64), n_points)
sparse_slice <- NeuroSlice(sparse_data, slice_space, indices = sparse_indices)
```

`NeuroSlice-class` *NeuroSlice Class*

Description

Represents a two-dimensional brain image slice. This class extends both the `array` class for data storage and the NeuroObj class for spatial information.

Details

NeuroSlice objects are typically used to represent individual slices of 3D brain volumes or 2D projections of 3D data. They inherit the spatial properties from NeuroObj and the data storage capabilities from array.

See Also

NeuroObj-class, NeuroVol-class

Examples

```
# Create a simple 2D brain slice
slice_data <- matrix(rnorm(64*64), 64, 64)
slice_space <- NeuroSpace(dim=c(64L, 64L), origin=c(0, 0), spacing=c(1, 1))
brain_slice <- new("NeuroSlice", .Data=slice_data, space=slice_space)
```

`NeuroSpace` *NeuroSpace: Spatial Reference System for Neuroimaging Data*

Description

The `NeuroSpace` class defines the spatial properties and coordinate system of neuroimaging data. It encapsulates all information needed to map between voxel indices and real-world coordinates, including dimensions, voxel spacing, origin, axis orientation, and coordinate transformations.

Usage

```
NeuroSpace(dim, spacing = NULL, origin = NULL, axes = NULL, trans = NULL)
```

Arguments

`dim`	An integer vector specifying the dimensions of the image grid. Must be positive.
`spacing`	A numeric vector specifying the physical size of each voxel (typically in millimeters). Must be positive. If NULL, defaults to ones.
`origin`	A numeric vector specifying the real-world coordinates of the first voxel. If NULL, defaults to zeros.
`axes`	An `AxisSet` object defining the orientation and ordering of the coordinate axes. If NULL, defaults to standard neurological convention (Left-Posterior-Inferior for 3D).
`trans`	A transformation matrix mapping voxel indices to world coordinates. If NULL, constructed from spacing and origin.

Details

Spatial Reference System for Neuroimaging Data

Value

A new `NeuroSpace` object

Coordinate Systems

NeuroSpace manages two coordinate systems:

- Voxel coordinates: Zero-based indices into the image grid
- World coordinates: Real-world coordinates (typically in millimeters)

The transformation between these systems is defined by:

- Voxel spacing (physical size of voxels)
- Origin (world coordinates of first voxel)
- Axis orientation (how image axes map to anatomical directions)

Validation

The constructor performs extensive validation:

- All dimensions must be positive integers
- All spacing values must be positive
- Origin and spacing must have matching lengths
- Transformation matrix must be invertible

References

For details on neuroimaging coordinate systems:

- Brett, M., Johnsrude, I. S., & Owen, A. M. (2002). The problem of functional localization in the human brain. Nature Reviews Neuroscience, 3(3), 243-249.
- Evans, A. C., et al. (1993). 3D statistical neuroanatomical models from 305 MRI volumes. Nuclear Science Symposium and Medical Imaging Conference.

See Also

`AxisSet` for axis orientation specification, `coord_to_index` for coordinate conversion, `index_to_coord` for inverse coordinate conversion, `NeuroObj` for objects using NeuroSpace

Examples

```
# Create a standard 3D space (64x64x40 voxels, 2mm isotropic)
space_3d <- NeuroSpace(
  dim = c(64L, 64L, 40L),
  spacing = c(2, 2, 2),
  origin = c(-90, -126, -72)
)

# Check properties
dim(space_3d)          # Image dimensions
spacing(space_3d)      # Voxel sizes
origin(space_3d)       # World-space origin

# Create a 2D slice space
space_2d <- NeuroSpace(
  dim = c(128L, 128L),
  spacing = c(1.5, 1.5),
  origin = c(-96, -96)
)

# Convert between coordinate systems
world_coords <- c(0, 0, 0)
vox_idx <- coord_to_index(space_3d, world_coords)
back_to_world <- index_to_coord(space_3d, vox_idx)
```

`NeuroSpace-class` *NeuroSpace Class*

Description

The NeuroSpace class represents the geometric properties of a brain image, including its dimensions, origin, spacing, axes, and coordinate transformations. It provides a comprehensive framework for handling spatial information in neuroimaging data analysis.

Slots

`dim` An integer vector representing the grid dimensions of the image.

`origin` A numeric vector representing the coordinates of the spatial origin.

`spacing` A numeric vector representing the dimensions (in mm) of the grid units (voxels).

`axes` A named `AxisSet` object representing the set of spatial axes in the untransformed native grid space.

`trans` A matrix representing an affine transformation that converts grid coordinates to real-world coordinates.

`inverse` A matrix representing an inverse transformation that converts real-world coordinates to grid coordinates.

Validity

A `NeuroSpace` object is considered valid if:

- The length of the `dim` slot is equal to the lengths of the `spacing`, `origin`, and number of axes in the `axes` slots.
- The `dim` slot contains only non-negative values.

Methods

The following methods are available for `NeuroSpace` objects:

- `dim`: Get the dimensions of the space.
- `origin`: Get or set the origin of the space.
- `spacing`: Get or set the spacing of the space.
- `axes`: Get the axes of the space.
- `trans`: Apply the affine transformation to coordinates.

Usage

The `NeuroSpace` class is fundamental in representing and manipulating the spatial properties of neuroimaging data. It is used extensively throughout the package for operations that require spatial information, such as image registration, resampling, and coordinate transformations.

References

For more information on spatial transformations in neuroimaging: Brett, M., Johnsrude, I. S., & Owen, A. M. (2002). The problem of functional localization in the human brain. Nature Reviews Neuroscience, 3(3), 243-249.

See Also

`AxisSet-class` for details on the axis set representation. `NeuroVol-class` and `NeuroVec-class` for classes that use NeuroSpace.

Examples

```
# Create a NeuroSpace object
space <- NeuroSpace(dim = c(64L, 64L, 64L),
                    origin = c(0, 0, 0),
                    spacing = c(1, 1, 1))

# Get the dimensions
dim(space)
```

`NeuroVec-class` *NeuroVec Class*

Description

This S4 class represents a four-dimensional brain image, which is used to store and process time series neuroimaging data such as fMRI or 4D functional connectivity maps. The class extends the basic functionality of `NeuroObj`.

The `NeuroVec` class represents a vectorized form of neuroimaging data, supporting both in-memory and file-backed data modes. It provides efficient data storage and access methods and integrates with the spatial reference system provided by `NeuroSpace`.

Usage

```
NeuroVec(data, space = NULL, mask = NULL, label = "")
```

Arguments

`data` The image data. This can be:

- A matrix (voxels x time points)
- A 4D array
- A list of `NeuroVol` objects

If a list of NeuroVol objects is provided, the geometric space (`NeuroSpace`) will be inferred from the constituent volumes, which must all be identical.

`space`	An optional `NeuroSpace` object defining the spatial properties of the image. Not required if `data` is a list of NeuroVol objects.
`mask`	An optional logical array specifying which voxels to include. If provided, a SparseNeuroVec object will be created.
`label`	A character string providing a label for the NeuroVec object. Default is an empty string.

Details

NeuroVec objects are designed to handle 4D neuroimaging data, where the first three dimensions represent spatial coordinates, and the fourth dimension typically represents time or another series dimension. This structure is particularly useful for storing and analyzing functional MRI data, time series of brain states, or multiple 3D volumes in a single object.

The function performs several operations:

- If `data` is a list of NeuroVol objects, it combines them into a single 4D array.
- It checks that the dimensions of `data` match the provided `space`.
- Depending on whether a `mask` is provided, it creates either a DenseNeuroVec or a SparseNeuroVec object.

Value

A concrete instance of the `NeuroVec` class:

- If `mask` is provided: a `SparseNeuroVec` object
- Otherwise: a `DenseNeuroVec` object

Slots

space A `NeuroSpace` object defining the spatial properties of the image.

label A character string providing a label for the NeuroVec object.

Methods

Methods specific to NeuroVec objects may include operations for time series analysis, 4D data manipulation, and extraction of 3D volumes or time courses.

Usage

To create a NeuroVec object, use the constructor function `NeuroVec()`. This function should handle the appropriate initialization of the 4D data structure and associated spatial information.

See Also

`NeuroObj-class` for the parent class. `DenseNeuroVec-class` and `SparseNeuroVec-class` for specific implementations.

`NeuroSpace` for spatial information, `sub_vector` for subsetting routines, and `index_to_coord` for coordinate conversion. `DenseNeuroVec-class`, `SparseNeuroVec-class` for the specific NeuroVec types. `NeuroVol-class` for 3D volumetric data.

Examples

```
# Load an example 4D brain image
example_4d_image <- read_vec(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))

# Create a NeuroVec object
neuro_vec <- NeuroVec(data = array(rnorm(64*64*32*10), dim = c(64, 64, 32, 10)),
                      space = NeuroSpace(dim = c(64, 64, 32,10),
                      origin = c(0, 0, 0),
                      spacing = c(3, 3, 4)))


dim(neuro_vec)

# Extract a single 3D volume (e.g., the first time point)
first_volume <- neuro_vec[[1]]


# Load an example 4D brain image
example_file <- system.file("extdata", "global_mask_v4.nii", package = "neuroim2")
example_4d_image <- read_vec(example_file)

# Create a DenseNeuroVec object
dense_vec <- NeuroVec(data = example_4d_image@.Data,
                      space = space(example_4d_image))
print(dense_vec)

# Create a SparseNeuroVec object with a mask
mask <- array(runif(prod(dim(example_4d_image)[1:3])) > 0.5,
              dim = dim(example_4d_image)[1:3])
sparse_vec <- NeuroVec(data = example_4d_image@.Data,
                       space = space(example_4d_image),
                       mask = mask)
print(sparse_vec)
```

`NeuroVecSeq` *NeuroVecSeq: A Container for Sequential NeuroVec Objects*

Description

The NeuroVecSeq class provides a container for managing a sequence of NeuroVec objects, particularly useful for handling time series or multi-session neuroimaging data where each segment may have different lengths.

Constructs a NeuroVecSeq object to represent a variable-length sequence of NeuroVec objects. This is particularly useful for managing time series data where different segments may have different lengths.

Usage

```
NeuroVecSeq(...)
```

Arguments

`...`	One or more instances of type `NeuroVec`.

Details

NeuroVecSeq objects store:

- A list of NeuroVec objects, each potentially with different time dimensions
- The lengths of each constituent NeuroVec
- A combined NeuroSpace object representing the total space

The class provides methods for:

- Accessing individual time points across all vectors
- Extracting subsequences
- Computing statistics across the sequence
- Linear access to the underlying data

The function performs several validations:

- Ensures all inputs are NeuroVec objects
- Verifies spatial compatibility
- Combines spatial information appropriately

Value

A NeuroVecSeq object containing:

- The provided NeuroVec objects
- Associated space information
- Length information for each vector

Methods

[[Extract a single volume at a specified time point

length Get the total number of time points

sub_vector Extract a subsequence of volumes

linear_access Access data linearly across all vectors

See Also

`NeuroVec` for the base vector class, `NeuroSpace` for spatial information

Examples

```
# Create some example NeuroVec objects
v1 <- NeuroVec(array(0, c(5, 5, 5, 2)),
               space = NeuroSpace(dim = c(5, 5, 5, 2)))
v2 <- NeuroVec(array(1, c(5, 5, 5, 4)),
               space = NeuroSpace(dim = c(5, 5, 5, 4)))
v3 <- NeuroVec(array(2, c(5, 5, 5, 6)),
               space = NeuroSpace(dim = c(5, 5, 5, 6)))

# Combine them into a sequence
vs <- NeuroVecSeq(v1, v2, v3)

# Access properties
length(vs)  # Total time points
vs[[5]]     # Get the 5th volume

# Extract a subsequence
sub_seq <- sub_vector(vs, 1:5)


# Create sample vectors
v1 <- NeuroVec(array(0, c(5, 5, 5, 2)),
               space = NeuroSpace(dim = c(5, 5, 5, 2)))
v2 <- NeuroVec(array(0, c(5, 5, 5, 4)),
               space = NeuroSpace(dim = c(5, 5, 5, 4)))

# Combine into sequence
vs <- NeuroVecSeq(v1, v2)
print(vs)
```

`NeuroVecSeq-class` *NeuroVecSeq Class*

Description

A concatenated sequence of `NeuroVec` instances.

Slots

`vecs` The sequences of `NeuroVec` instances

`lens` The number of volumes in each `NeuroVec` sequence

`NeuroVecSource` *NeuroVecSource*

Description

This function constructs a NeuroVecSource object, which represents the source of a four-dimensional brain image.

Usage

```
NeuroVecSource(file_name, indices = NULL, mask = NULL)
```

Arguments

`file_name`	The name of the 4-dimensional image file.
`indices`	An optional integer vector specifying the subset of volume indices to load. If not provided, all volumes will be loaded.
`mask`	An optional logical array or `NeuroVol` object defining the subset of voxels to load. If provided, a SparseNeuroVecSource object will be created.

Details

If a `mask` is supplied, it should be a `LogicalNeuroVol` or `NeuroVol` instance. If the latter, then the mask will be defined by nonzero elements of the volume.

Value

An instance of the `NeuroVecSource` class.

`NeuroVecSource-class` *NeuroVecSource Class*

Description

A class used to produce a `NeuroVec` instance.

Slots

`indices` An `integer` vector representing the indices of the volumes to be loaded.

See Also

`FileSource-class`, `NeuroVec-class`

NeuroVol *NeuroVol: 3D Neuroimaging Volume Class*

Description

The `NeuroVol` class encapsulates 3D volumetric neuroimaging data. It provides methods for accessing slices, performing spatial transformations, and integrating with the spatial reference provided by `NeuroSpace`.

Usage

```
NeuroVol(data, space, label = "", indices = NULL)
```

Arguments

`data`	A 3D array containing the volumetric data.
`space`	An object of class `NeuroSpace` defining the spatial properties.
`label`	A character string providing a label for the volume (default: "").
`indices`	An optional vector of indices for sparse representation (default: NULL).

Value

A `NeuroVol` object.

Examples

```
bspace <- NeuroSpace(c(64,64,64), spacing=c(1,1,1))
dat <- array(rnorm(64*64*64), c(64,64,64))
bvol <- NeuroVol(dat,bspace, label="test")
```

NeuroVol-class *NeuroVol Class*

Description

Base class for representing 3D volumetric neuroimaging data. This class extends `NeuroObj` to provide a foundation for various types of 3D brain images.

Details

NeuroVol serves as an abstract base class for more specific 3D neuroimaging data structures. It inherits spatial properties from NeuroObj but does not specify a particular data storage method.

See Also

`NeuroObj-class`, `DenseNeuroVol-class`

`NeuroVolSource` *Constructor for NeuroVolSource*

Description

Constructor for NeuroVolSource

Usage

```
NeuroVolSource(input, index = 1)
```

Arguments

`input`	the input file name
`index`	the image subvolume index

Value

a new instance of type `NeuroVolSource`

`NiftiExtension` *Create a NIfTI Extension*

Description

Constructor function for creating a `NiftiExtension-class` object with proper padding to ensure the size is a multiple of 16 bytes.

Usage

```
NiftiExtension(ecode, data)
```

Arguments

`ecode`	Integer extension code. See `NiftiExtensionCodes` for known codes. Common values: 4 (AFNI), 6 (comment), 32 (CIFTI).
`data`	The extension data. Can be: • A character string (will be converted to raw with null terminator) • A raw vector (used as-is)

Details

The function automatically handles padding to ensure the total extension size (esize) is a multiple of 16 bytes, as required by the NIfTI specification. The esize includes the 8-byte header (esize + ecode fields).

Value

A `NiftiExtension-class` object.

See Also

`NiftiExtension-class`, `NiftiExtensionCodes`

Examples

```
# Create a comment extension
ext <- NiftiExtension(ecode = 6L, data = "This is a comment")
ext@ecode
ext@esize

# Create an AFNI extension with XML data
afni_xml <- '<?xml version="1.0"?><AFNI_attributes></AFNI_attributes>'
afni_ext <- NiftiExtension(ecode = 4L, data = afni_xml)
```

`NiftiExtension-class` *NiftiExtension Class*

Description

Represents a single NIfTI header extension block. NIfTI extensions allow additional metadata to be stored with the image file.

Usage

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

Arguments

`object`	A `NiftiExtension` object.

Details

NIfTI-1.1 extensions follow this structure:

- Bytes 0-3: esize (int32) - total extension size, must be multiple of 16
- Bytes 4-7: ecode (int32) - extension code identifying format
- Bytes 8-(esize-1): edata - the actual extension data

Extensions are chained sequentially after the NIfTI header (byte 352) until the vox_offset is reached.

Slots

`ecode` An `integer` extension code identifying the type of extension. See `NiftiExtensionCodes` for known codes.

`esize` An `integer` giving the total size of the extension in bytes, including the 8-byte header (esize + ecode). Must be a multiple of 16.

`edata` A `raw` vector containing the extension data (length = esize - 8).

See Also

`NiftiExtensionCodes` for registered extension codes. `NiftiExtensionList-class` for a collection of extensions. `parse_extension` for parsing extension data.

Examples

```
# Create a simple comment extension
comment_text <- "This is a test comment"
ext <- NiftiExtension(ecode = 6L, data = comment_text)

# Access the extension code
ext@ecode
```

`NiftiExtensionCodes` *Known NIfTI Extension Codes*

Description

A named integer vector of registered NIfTI extension codes. These codes identify the format/type of extension data.

Usage

```
NiftiExtensionCodes
```

Format

Named integer vector where names describe the extension type:

ignore 0 - Unknown/private format (not recommended)

DICOM 2 - DICOM format (attribute tags and values)

AFNI 4 - AFNI group (ASCII XML attributes)

comment 6 - Plain text comment

XCEDE 8 - XCEDE format

jimdiminfo 10 - JIM dimension info

workflow_fwds 12 - Workflow forwards

FreeSurfer 14 - FreeSurfer format

pypickle 16 - Python pickle

MiND_ident 18 - MiND identifier

b_value 20 - B-value (diffusion)

spherical_direction 22 - Spherical direction

DT_component 24 - DT component

SHC_degreeorder 26 - SHC degree order

voxbo 28 - VoxBo format

Caret 30 - Caret format

CIFTI 32 - CIFTI format

variable_frame_timing 34 - Variable frame timing

eval 38 - Eval

MATLAB 40 - MATLAB format

Quantiphyse 42 - Quantiphyse

MRS 44 - MRS NIfTI

Examples

```
# Get the code for AFNI extensions
NiftiExtensionCodes["AFNI"]  # Returns 4

# Get the name for a code
names(NiftiExtensionCodes)[NiftiExtensionCodes == 4]  # Returns "AFNI"
```

`NiftiExtensionList-class`

NiftiExtensionList Class

Description

A validated list containing zero or more `NiftiExtension-class` objects. This class ensures type safety when working with collections of NIfTI extensions.

Usage

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

Arguments

`object`	A `NiftiExtensionList` object.

Details

The class extends `list` and enforces that all elements must be `NiftiExtension` objects. This provides a clean container for managing multiple extensions attached to a NIfTI file.

See Also

`NiftiExtension-class` for individual extension objects. `extensions` for accessing extensions from image objects.

Examples

```
# Create an empty extension list
ext_list <- new("NiftiExtensionList")

# Create a list with extensions
ext1 <- NiftiExtension(ecode = 6L, data = "Comment 1")
ext2 <- NiftiExtension(ecode = 6L, data = "Comment 2")
ext_list <- new("NiftiExtensionList", list(ext1, ext2))
```

`NIFTIMetaInfo` *Create NIFTI Format Metadata Object*

Description

Creates a NIFTIMetaInfo object containing format-specific metadata for NIFTI format neuroimaging files.

Usage

```
NIFTIMetaInfo(descriptor, nifti_header)
```

Arguments

`descriptor`	NIFTIFormat object specifying file format details
`nifti_header`	List containing NIFTI header information

Details

Create NIFTIMetaInfo Object

The NIFTIMetaInfo object extends MetaInfo with NIFTI-specific features:

- NIFTI header fields (qform, sform matrices)
- Data scaling (slope, intercept)
- File organization (separate vs. single file)
- Orientation information

Validation ensures:

- Valid NIFTI format
- Consistent dimensions
- Valid transformation matrices
- Proper data scaling

Value

A NIFTIMetaInfo object

See Also

`MetaInfo`

Examples

```
# Read NIFTI header
header <- read_header(system.file("extdata", "global_mask_v4.nii", package="neuroim2"))

# Create format descriptor
fmt <- new("NIFTIFormat",
           file_format = "NIFTI",
           header_encoding = "raw",
           header_extension = "nii",
           data_encoding = "raw",
           data_extension = "nii")

# Create metadata
meta <- NIFTIMetaInfo(fmt, header@header)

# Check dimensions
dim(meta)
```

`None` *Pre-defined null axis*

Description

Pre-defined null axis

Usage

```
None
```

Format

An object of class `NamedAxis` of length 1.

`NullAxis` *Pre-defined null axis set*

Description

Pre-defined null axis set

Usage

```
NullAxis
```

Format

An object of class `AxisSet` of length 1.

`numericOrMatrix-class` *numericOrMatrix Union*

Description

A class union that includes both numeric vectors and matrices.

`num_clusters` *Number of Clusters*

Description

This function returns the number of clusters in a ClusteredNeuroVol object.

Usage

```
num_clusters(x)

## S4 method for signature 'ClusteredNeuroVec'
num_clusters(x)

## S4 method for signature 'ClusteredNeuroVol'
num_clusters(x)
```

Arguments

`x`	A ClusteredNeuroVol object.

Value

An `integer` representing the number of clusters in `x`.

An integer representing the number of clusters in the input object.

Examples

```
# Create a simple 3D volume and mask
space <- NeuroSpace(c(16, 16, 16), spacing = c(1, 1, 1))
vol_data <- array(rnorm(16^3), dim = c(16, 16, 16))
mask_vol <- LogicalNeuroVol(vol_data > 0, space)

# Get coordinates of masked voxels for clustering
mask_idx <- which(mask_vol)
coords <- index_to_coord(mask_vol, mask_idx)

# Cluster the coordinates into 10 groups using k-means
set.seed(123)  # for reproducibility
kmeans_result <- kmeans(coords, centers = 10)

# Create a clustered volume
clustered_vol <- ClusteredNeuroVol(mask_vol, kmeans_result$cluster)

# Get the number of clusters
n_clusters <- num_clusters(clustered_vol)
n_clusters == 10
```

`OrientationList2D` *Pre-defined 2D orientation configurations*

Description

A list of standard 2D anatomical orientations used in neuroimaging. Each orientation defines a pair of anatomical axes.

Usage

```
OrientationList2D
```

Format

An object of class `list` of length 24.

`OrientationList3D` *Pre-defined 3D orientation configurations*

Description

A list of standard 3D anatomical orientations used in neuroimaging. Each orientation defines a triplet of anatomical axes.

Usage

```
OrientationList3D
```

Format

An object of class `list` of length 48.

`origin` *Extract Image Origin*

Description

Extract Image Origin

Usage

```
origin(x)

## S4 method for signature 'NeuroSpace'
origin(x)

## S4 method for signature 'NeuroVol'
origin(x)

## S4 method for signature 'NeuroVec'
origin(x)
```

Arguments

`x`	an object with an origin

Value

A numeric vector giving the origin of `x`.

Examples

```
bspace <- NeuroSpace(c(10,10,10), c(2,2,2))
stopifnot(origin(bspace) == c(0,0,0))
```

`parse_afni_extension` *Parse AFNI Extension*

Description

Parses an AFNI extension (ecode = 4) containing XML-formatted attributes.

Usage

```
parse_afni_extension(ext, as_xml = TRUE)
```

Arguments

`ext`	A `NiftiExtension-class` object with ecode = 4.
`as_xml`	Logical; if TRUE (default) and xml2 is available, returns an xml_document object. Otherwise returns the raw XML string.

Details

AFNI stores dataset attributes in an XML format within the NIfTI extension. The XML contains elements like HISTORY_NOTE, volume labels, tagged points, and other AFNI-specific metadata.

Value

If `as_xml` = `TRUE` and xml2 is available, returns an xml_document. Otherwise returns a character string containing the XML.

See Also

`get_afni_attribute` for extracting specific AFNI attributes.

Examples

```
## Not run:
# Read a NIfTI file with AFNI extension
hdr <- read_nifti_header("afni_file.nii")
afni_ext <- hdr$extensions[[1]]
parsed <- parse_afni_extension(afni_ext)

## End(Not run)
```

`parse_extension` *Parse NIfTI Extension Data*

Description

Parses the raw data in a NIfTI extension based on its extension code. Provides specialized parsing for known extension types.

Usage

```
parse_extension(ext, ...)
```

Arguments

`ext`	A `NiftiExtension-class` object.
`...`	Additional arguments passed to type-specific parsers.

Value

Parsed data in an appropriate format:

- ecode 4 (AFNI): An XML document (if xml2 available) or character string
- ecode 6 (comment): Character string
- Other codes: Raw vector (unchanged)

See Also

`parse_afni_extension` for AFNI-specific parsing.

Examples

```
# Parse a comment extension
ext <- NiftiExtension(ecode = 6L, data = "Test comment")
parse_extension(ext)  # Returns "Test comment"
```

`partition` *Partition an image into a set of disjoint clusters*

Description

This function partitions an image into a set of disjoint clusters using k-means clustering.

Usage

```
partition(x, k, ...)

## S4 method for signature 'LogicalNeuroVol,integer'
partition(x, k)

## S4 method for signature 'LogicalNeuroVol,numeric'
partition(x, k)

## S4 method for signature 'DenseNeuroVol,numeric'
partition(x, k)
```

Arguments

`x`	the image to partition, represented as a 3D array.
`k`	the number of clusters to form.
`...`	additional arguments passed to the kmeans function.

Value

a 3D `array` where each voxel is assigned to a cluster.

See Also

`kmeans`

Examples

```
# Load an example 3D image
library(neuroim2)
img <- read_vol(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))

# Partition the image into 5 clusters using default options
clusters <- partition(img, 5)
```

`patch_set` *Generate a set of coordinate "patches" of fixed size from an image object.*

Description

Generate a set of coordinate "patches" of fixed size from an image object.

Usage

```
patch_set(x, dims, mask, ...)
```

Arguments

`x`	the object to extract patches from
`dims`	a vector indicating the dimensions of the patches
`mask`	mask indicating the valid patch area
`...`	additional args

Value

A `list` of coordinate patches, each representing a fixed-size region of the input object.

Examples

```
# Create a simple 3D volume
space <- NeuroSpace(c(10,10,10), spacing=c(1,1,1))
vol <- NeuroVol(array(rnorm(1000), c(10,10,10)), space)

# Create a mask with some active voxels
mask <- LogicalNeuroVol(vol > 0, space)

# Extract 3x3x3 patches centered at each active voxel
patches <- patch_set(vol, dims=c(3,3,3), mask=mask)

# Access the first patch
patch1 <- patches[[1]]
dim(patch1)  # Should be c(27) (flattened 3x3x3 patch)
```

`patch_set,NeuroVol,numeric,missing-method` *Create a patch set from a NeuroVol object*

Description

This function creates a patch set from a NeuroVol object given specified dimensions

This function creates a patch set from a NeuroVol object given specified dimensions and a mask.

Usage

```
## S4 method for signature 'NeuroVol,numeric,missing'
patch_set(x, dims, mask, ...)

## S4 method for signature 'NeuroVol,numeric,LogicalNeuroVol'
patch_set(x, dims, mask, ...)
```

Arguments

`x`	a NeuroVol object
`dims`	the dimensions of the patch
`mask`	the mask defining the valid patch centers
`...`	additional args

Value

A deferred list of patches.

A deferred list of patches.

`perm_mat` *Extract permutation matrix associated with an image*

Description

A permutation matrix defines how the native voxel coordinates can be transformed to standard (LPI) orientation.

Usage

```
perm_mat(x, ...)

## S4 method for signature 'NeuroSpace'
perm_mat(x, ...)
```

Arguments

`x`	A NeuroSpace object
`...`	Additional arguments (not used)

Details

a permutation matrix can be used to convert between cardinal image orientations. For example, if an image is stored in "RPI" (Right-Posterior-Inferior) format, a coordinate in this space can be converted to LPI (Left-Posterior-Inferior) by multiplying a coordinate vector by the permutation matrix.

Value

A numeric N x N `matrix` representing the permutation transform, where N is the dimensionality of the image.

A numeric N x N `matrix` representing the permutation transform, where N is the dimensionality of the image.

Examples

```
fname <- system.file("extdata", "global_mask_v4.nii", package="neuroim2")
vol <- read_vol(fname)
pmat <- perm_mat(space(vol))

vox <- c(12,12,8)
pvox <- vox %*% perm_mat(space(vol))

stopifnot(all(pvox == c(-12,12,8)))
```

`perm_mat,AxisSet2D-method` *Get permutation matrix from axis set*

Description

Get permutation matrix from axis set

Usage

```
## S4 method for signature 'AxisSet2D'
perm_mat(x, ...)
```

Arguments

`x`	An AxisSet2D object
`...`	Additional arguments (not used)

Value

A matrix representing the axis directions

---

perm_mat,AxisSet3D-method

Get permutation matrix from axis set

Description

Get permutation matrix from axis set

Usage

```
## S4 method for signature 'AxisSet3D'
perm_mat(x, ...)
```

Arguments

`x`	An AxisSet3D object
`...`	Additional arguments (not used)

Value

A matrix representing the axis directions

---

plot,NeuroSlice-method

Plot a NeuroSlice

Description

Plot a NeuroSlice

plot a NeuroVol

Usage

```
## S4 method for signature 'NeuroSlice'
plot(
  x,
  cmap = gray(seq(0, 1, length.out = 255)),
  irange = range(x, na.rm = TRUE),
  legend = TRUE
)

## S4 method for signature 'NeuroVol'
```

```
plot(
  x,
  cmap = gray(seq(0, 1, length.out = 255)),
  zlevels = unique(round(seq(1, dim(x)[3], length.out = 6))),
  irange = range(x, na.rm = TRUE),
  thresh = c(0, 0),
  alpha = 1,
  bgvol = NULL,
  bgcmap = gray(seq(0, 1, length.out = 255)),
  legend = TRUE
)
```

Arguments

`x`	the object to display
`cmap`	a color map consisting of a vector of colors in hex format (e.g. `gray(n=255)`)
`irange`	the intensity range indicating the low and high values of the color scale.
`legend`	Logical indicating whether to display the color legend. Defaults to TRUE.
`zlevels`	the series of slice indices to display.
`thresh`	a 2-element vector indicating the lower and upper transparency thresholds.
`alpha`	the level of alpha transparency
`bgvol`	a background volume that serves as an image underlay (currently ignored).
`bgcmap`	a color map for backround layer consisting of a vector of colors in hex format (e.g. `gray(n=255)`)

Details

The plot method uses `ggplot2` to create a raster visualization of the slice data. The intensity values are mapped to colors using the specified colormap and range.

when ‘x‘ is a NeuroSlice object, the plot method returns a `ggplot2` object containing the raster visualization of the slice data. The plot can be further customized using standard ggplot2 functions.

Value

a ggplot2 object

Examples

```
# Create example slice
slice_space <- NeuroSpace(c(100, 100))
slice_data <- matrix(rnorm(100*100), 100, 100)
slice <- NeuroSlice(slice_data, slice_space)

# Basic plot
plot(slice)
```

```
dat <- matrix(rnorm(100*100), 100, 100)
slice <- NeuroSlice(dat, NeuroSpace(c(100,100)))

plot(slice)
```

`plot_montage` *Plot a montage of axial (or any-plane) slices using facetting*

Description

This avoids extra dependencies by using a single ggplot with facets and a shared colorbar. Supply a list of slice objects or a volume + indices.

Usage

```
plot_montage(
  x,
  zlevels = NULL,
  along = 3L,
  cmap = "grays",
  range = c("robust", "data"),
  probs = c(0.02, 0.98),
  ncol = 6L,
  downsample = 1L,
  title = NULL,
  subtitle = NULL,
  caption = NULL
)
```

Arguments

`x`	Either a 3D volume object accepted by ‘slice()‘ or a list of slices.
`zlevels`	Integer indices of slices to plot (if ‘x‘ is a volume).
`along`	Axis along which to slice (1 = sagittal, 2 = coronal, 3 = axial).
`cmap`	Palette name or vector (see [resolve_cmap()]).
`range`	"robust" (quantile-based) or "data" (min/max).
`probs`	Quantiles for ‘range="robust"‘.
`ncol`	Number of columns in the facet layout.
`downsample`	Integer decimation for speed.
`title`, `subtitle`, `caption`	Optional ggplot labels.

`plot_ortho` *Orthogonal three-plane view with optional crosshairs*

Description

Creates axial, coronal, and sagittal panels at a given coordinate with harmonized aesthetics. Returns (invisibly) the three ggplot objects after printing them in a single row using base grid (no extra deps).

Usage

```
plot_ortho(
  vol,
  coord = NULL,
  unit = c("index", "mm"),
  cmap = "grays",
  range = c("robust", "data"),
  probs = c(0.02, 0.98),
  crosshair = TRUE,
  annotate = TRUE,
  downsample = 1L
)
```

Arguments

`vol`	A 3D volume handled by ‘slice()‘.
`coord`	Length-3 coordinate of the target point. Interpreted as voxel indices by default; set ‘unit = "mm"‘ to convert using ‘coord_to_grid()‘ if available in your environment.
`unit`	"index" or "mm".
`cmap`	Palette for the slices.
`range`	"robust" or "data" for intensity limits shared by all panels.
`probs`	Quantiles for robust range.
`crosshair`	Logical; draw crosshair lines.
`annotate`	Logical; add orientation glyphs.
`downsample`	Integer decimation for speed.

`plot_overlay` *Composite an overlay map on a structural background*

Description

Works without extra packages by colorizing both layers to rasters and stacking them as grobs. Great for statistical maps over T1/T2 backgrounds.

Usage

```
plot_overlay(
  bgvol,
  overlay,
  zlevels = NULL,
  along = 3L,
  bg_cmap = "grays",
  ov_cmap = "inferno",
  bg_range = c("robust", "data"),
  ov_range = c("robust", "data"),
  probs = c(0.02, 0.98),
  ov_thresh = 0,
  ov_alpha = 0.7,
  ncol = 3L,
  title = NULL,
  subtitle = NULL,
  caption = NULL
)
```

Arguments

`bgvol`	Background 3D volume.
`overlay`	Overlay 3D volume (same dims as bgvol).
`zlevels`	Slices to plot (indices along the z/3rd axis by default).
`along`	Axis for slicing (1 sagittal, 2 coronal, 3 axial).
`bg_cmap`	Background palette (e.g., "grays").
`ov_cmap`	Overlay palette (e.g., "inferno").
`bg_range`, `ov_range`	"robust" or "data" for background/overlay scaling.
`probs`	Quantiles for robust scaling.
`ov_thresh`	Numeric threshold; values with |v| < thresh become transparent.
`ov_alpha`	Global alpha for overlay (0..1).
`ncol`	Number of columns in the facet layout.
`title`, `subtitle`, `caption`	Optional labels.

`prepare_confounds` *Prepare weighted nuisance projectors for each run*

Description

Converts per-run confound matrices and time weights into orthonormal projectors that can be consumed directly by the C++ graph builder. Each run produces Q_k (columns spanning the weighted confound space) and $\sqrt{w_k}$ (per-time-point square-root weights). Supplying a `NULL` confound matrix yields a zero-column projector, enabling pure time-weighting without regression.

Usage

```
prepare_confounds(
  confounds,
  time_weights = NULL,
  run_lengths,
  include_intercept = TRUE,
  center_cols = TRUE,
  scale_cols = FALSE
)
```

Arguments

`confounds`	List of matrices ($T_k \times p_k$), or a single matrix reused for all runs. Each row corresponds to a time point.
`time_weights`	Optional list of numeric vectors (length T_k) or a single vector reused for every run. If `NULL`, unit weights are used.
`run_lengths`	Integer vector with the number of time points per run. Required when any run has both `confounds=NULL` and `time_weights=NULL`.
`include_intercept`	Logical; prepend a column of ones before QR (default TRUE).
`center_cols`	Logical; center each confound column before weighting (default TRUE).
`scale_cols`	Logical; scale columns to unit variance before weighting (default FALSE).

Value

A list with elements `Q_list` (list of matrices) and `sqrtw_list` (list of numeric vectors). Each entry has the same length as `run_lengths`.

`quaternToMatrix` *Convert Quaternion Parameters to a Transformation Matrix*

Description

Given a quaternion `(b, c, d)`, a scalar offset (origin), voxel step sizes, and the `qfac` sign, reconstructs a 4x4 affine matrix representing rotation, scaling, and translation as used in NIfTI-1.

Usage

```
quaternToMatrix(quat, origin, stepSize, qfac)
```

Arguments

`quat`	A numeric vector of length 3 containing the quaternion parameters (b, c, d). The scalar part a is computed internally.
`origin`	A numeric vector of length 3 specifying the translation components (often the real-space origin or offset).
`stepSize`	A numeric vector of length 3 giving the voxel dimensions along each axis (e.g., `(dx, dy, dz)`).
`qfac`	Either `+1` or `-1`, indicating the sign from the determinant check in `matrixToQuatern`.

Details

This function uses the quaternion formalism common in neuroimaging, adding the offset (translation) into the 4th column, and applying the voxel sizes along each axis. If `qfac` is `-1`, the z scale is negated. The resulting 4x4 matrix is typically used as an affine transform for voxel-to-world coordinate mapping.

Value

A 4x4 numeric affine transformation matrix. The top-left 3x3 submatrix encodes rotation and scaling, and the 4th column encodes translation.

See Also

`matrixToQuatern` for converting a matrix back to quaternion form.

`random_searchlight` *Create a spherical random searchlight iterator*

Description

This function generates a spherical random searchlight iterator for analyzing local neighborhoods of voxels within a given radius in a brain mask.

Usage

```
random_searchlight(mask, radius, nonzero = TRUE)
```

Arguments

`mask`	A `NeuroVol` object representing the brain mask.
`radius`	A numeric value specifying the radius of the searchlight sphere in voxel units.
`nonzero`	Logical; if `TRUE` (default) discard zero-valued voxels in the mask when forming each searchlight.

Value

A list of `ROIVolWindow` objects, each representing a spherical searchlight region.

Examples

```
# Create a simple brain mask
mask_data <- array(TRUE, c(10, 10, 10))
mask_data[1, 1, 1] <- FALSE
mask <- LogicalNeuroVol(mask_data, NeuroSpace(c(10,10,10)))

# Generate random searchlight iterator with a radius of 2 voxels

searchlights <- random_searchlight(mask, radius = 6)
```

`read_elements,BinaryReader,numeric-method` *Read Elements from Binary Reader*

Description

Read a specified number of elements from a BinaryReader object.

Usage

```
## S4 method for signature 'BinaryReader,numeric'
read_elements(x, num_elements)
```

Arguments

`x`	Object of class BinaryReader
`num_elements`	Integer specifying number of elements to read

Value

Numeric vector of read elements

Examples

```
# Create a temporary binary file with some test data
tmp <- tempfile()
con <- file(tmp, "wb")
test_data <- rnorm(100)
writeBin(test_data, con, size = 8)
close(con)

# Create reader and read the data
reader <- BinaryReader(tmp, byte_offset = 0L,
                       data_type = "DOUBLE", bytes_per_element = 8L)
data <- read_elements(reader, 100)
close(reader)

# Clean up
unlink(tmp)
```

`read_header` *read header information of an image file*

Description

read header information of an image file

Usage

```
read_header(file_name)
```

Arguments

`file_name`	the name of the file to read

Value

an instance of class `FileMetaInfo`

Examples

```
hdr <- read_header(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))
dim(hdr)                  # image dimensions
hdr@header$pixdim[5]      # TR in seconds
```

`read_image` *read_image*

Description

Convenience wrapper that inspects the file metadata and dispatches to `read_vol` for 3D data or `read_vec` for 4D data.

Usage

```
read_image(
  file_name,
  type = c("auto", "vol", "vec"),
  index = 1,
  indices = NULL,
  mask = NULL,
  mode = c("normal", "mmap", "bigvec", "filebacked")
)
```

Arguments

`file_name`	Character vector of file paths.
`type`	One of `"auto"`, `"vol"`, or `"vec"` to override dispatch.
`index`	Volume index to use when returning a `NeuroVol` or when you want to load a subset of volumes while still returning a `NeuroVec`.
`indices`	Optional vector of indices passed through to `read_vec`.
`mask`	Optional mask passed to `read_vec`.
`mode`	IO mode forwarded to `read_vec`.

Value

A `NeuroVol` when the input is effectively 3D (or when `type = "vol"`), otherwise a `NeuroVec`/`NeuroVecSeq`.

Examples

```
vol <- read_image(system.file("extdata", "global_mask2.nii.gz", package = "neuroim2"))
vec <- read_image(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))
```

`read_meta_info` *Generic function to read image meta info given a file*

Description

Reads meta information from image files based on their format (NIFTI or AFNI).

Usage

```
read_meta_info(x, file_name)

## S4 method for signature 'NIFTIFormat'
read_meta_info(x, file_name)

## S4 method for signature 'AFNIFormat'
read_meta_info(x, file_name)
```

Arguments

`x`	A FileFormat object (either NIFTIFormat or AFNIFormat)
`file_name`	A character string specifying the file name to read meta information from

Details

These methods use format-specific functions to read the header information and create the appropriate meta information object. The ‘.read_meta_info‘ helper function is used internally to streamline the process for both formats.

Value

A `list` containing the meta information read from the file.

An object of class NIFTIMetaInfo or AFNIMetaInfo, depending on the input format

Examples

```
# Create a NIFTI format descriptor
fmt <- new("NIFTIFormat",
           file_format = "NIFTI",
           header_encoding = "raw",
           header_extension = "nii",
           data_encoding = "raw",
           data_extension = "nii")

# Read metadata from a NIFTI file

fname <- system.file("extdata", "global_mask_v4.nii", package="neuroim2")
meta <- read_meta_info(fmt, fname)
```

```
# Access metadata properties
dim(meta)          # Image dimensions
trans(meta)        # Transformation matrix
```

`read_vec` *read_vec*

Description

Loads a neuroimaging volume from one or more files, with support for various input formats and memory management strategies.

Usage

```
read_vec(
  file_name,
  indices = NULL,
  mask = NULL,
  mode = c("normal", "mmap", "bigvec", "filebacked")
)
```

Arguments

`file_name`	The name(s) of the file(s) to load. If multiple files are specified, they are loaded and concatenated along the time dimension.
`indices`	The indices of the sub-volumes to load (e.g. if the file is 4-dimensional). Only supported in "normal" mode.
`mask`	A logical mask defining which spatial elements to load. Required for "bigvec" mode and optional for other modes.
`mode`	The IO mode which is one of: * "normal": Standard in-memory loading * "mmap": Memory-mapped access (more memory efficient) * "bigvec": Optimized for large datasets with masking * "filebacked": File-backed storage with on-demand loading

Details

This function supports multiple file formats: * .nii: Standard NIfTI format * .nii.gz: Compressed NIfTI (not supported in mmap mode)

Memory management modes: * "normal": Loads entire dataset into memory. Best for smaller datasets or when memory is not a constraint. * "mmap": Memory-maps the file, providing efficient access for large files without loading entirely into memory. Not available for compressed files. * "bigvec": Optimized for large datasets where only a subset of voxels are of interest. Requires a mask to specify which voxels to load. * "filebacked": Similar to mmap but with more flexible caching strategies.

Value

An `NeuroVec` object representing the loaded volume(s).

Note

* Memory-mapping (.mmap mode) is not supported for gzipped files * For .lv.h5 and .h5 files, the indices and mask parameters are ignored * The bigvec mode requires a mask to be specified * When loading multiple files, they must have compatible dimensions

Examples

```
# Load a single NIfTI file
img <- read_vec(system.file("extdata", "global_mask_v4.nii", package="neuroim2"))


# Memory-mapped loading for large files
big_img <- read_vec(system.file("extdata", "global_mask_v4.nii", package="neuroim2"), mode="mmap")

# Load masked data for memory efficiency
mask <- as.logical(big_img[[1]])
masked_data <- read_vec(system.file("extdata", "global_mask_v4.nii", package="neuroim2"),
               mask=mask, mode="bigvec")
```

`read_vol` *Load an image volume from a file*

Description

Load an image volume from a file

Usage

```
read_vol(file_name, index = 1)
```

Arguments

`file_name`	the name of the file to load
`index`	the index of the volume (e.g. if the file is 4-dimensional)

Value

an instance of the class `DenseNeuroVol`

Examples

```
fname <- system.file("extdata", "global_mask_v4.nii", package="neuroim2")
x <- read_vol(fname)
print(dim(x))
space(x)
```

`read_vol_list` *read_vol_list*

Description

This function loads a list of image volumes and returns a NeuroVec object.

Usage

```
read_vol_list(file_names, mask = NULL)
```

Arguments

`file_names`	A list of file names to load.
`mask`	An optional mask defining the subset of voxels to load.

Value

An instance of the `NeuroVec` class.

`reorient` *Remap the grid-to-world coordinates mapping of an image.*

Description

Remap the grid-to-world coordinates mapping of an image.

Usage

```
reorient(x, orient)

## S4 method for signature 'NeuroSpace,character'
reorient(x, orient)
```

Arguments

`x`	the object
`orient`	the orientation code indicating the "remapped" axes.

Details

When `x` is a `NeuroSpace` object, the `orient` argument should be a character vector of length 3 specifying the desired anatomical orientation using single-letter codes. Each letter represents an anatomical direction:

- First position: "R" (Right) or "L" (Left)
- Second position: "A" (Anterior) or "P" (Posterior)
- Third position: "S" (Superior) or "I" (Inferior)

For example, `c("R", "A", "S")` specifies Right-Anterior-Superior orientation, while `c("L", "P", "I")` specifies Left-Posterior-Inferior orientation. The orientation codes determine how the voxel grid coordinates map to real-world anatomical space.

Value

A reoriented version of `x`.

Examples

```
# Create a NeuroSpace object in LPI (Left-Posterior-Inferior) orientation
space <- NeuroSpace(c(64, 64, 40), c(2, 2, 2))

# Reorient to RAS (Right-Anterior-Superior) orientation
# Use individual axis codes: "R" for Right, "A" for Anterior, "S" for Superior
space_ras <- reorient(space, c("R", "A", "S"))

# The transformation matrix will be updated to reflect the new orientation
# Original and reoriented spaces will have different coordinate mappings
coords <- c(32, 32, 20)
orig_world <- grid_to_coord(space, coords)
new_world <- grid_to_coord(space_ras, coords)
```

`resample` *Resample an Image to Match the Space of Another Image*

Description

This function resamples a source image to match the spatial properties (dimensions, resolution, and orientation) of a target image.

This method resamples a NeuroVol object (`source`) to match the dimensions and orientation of a NeuroSpace object (`target`).

This method preserves discrete cluster labels and label mappings when resampling clustered volumes to a new space.

Usage

```
resample(source, target, ...)

## S4 method for signature 'NeuroVol,NeuroVol'
resample(source, target, interpolation = 3L)

## S4 method for signature 'NeuroVol,NeuroSpace'
resample(source, target, interpolation = 3L)

## S4 method for signature 'ClusteredNeuroVol,NeuroSpace'
resample(source, target, interpolation = 0L)

## S4 method for signature 'ClusteredNeuroVol,NeuroVol'
resample(source, target, interpolation = 0L)
```

Arguments

`source`	A NeuroVol object representing the source volume to be resampled.
`target`	A NeuroSpace object representing the target space to match the dimensions and orientation of the source volume.
`...`	Additional arguments passed to the resampling function, such as interpolation method, boundary handling, or other resampling options.
`interpolation`	A single integer specifying the type of interpolation to be applied to the final resampled image. May be 0 (nearest neighbor), 1 (trilinear), or 3 (cubic spline). No other values are valid.

Value

An object representing the resampled `source` image, with the same spatial properties as `target`.

See Also

`NeuroVol` for the base volume class

Examples

```
img <- read_vol(system.file("extdata", "global_mask_v4.nii", package = "neuroim2"))
rspace <- space(img)


newtrans4X3 <- trans(img)[1:4, 1:3]
newtrans4X3 <- newtrans4X3 * c(.5,.5,.5,1)
newtrans <- cbind(newtrans4X3, c(space(img)@origin,1))

rspace <- NeuroSpace(rspace@dim*2, rspace@spacing/2, origin=rspace@origin, trans=trans(img))

rvol <- resample(img, rspace)
```

```
# Create source and target volumes
src_vol <- read_vol(system.file("extdata", "global_mask_v4.nii", package="neuroim2"))
targ_vol <- read_vol(system.file("extdata", "global_mask_v4.nii", package="neuroim2"))

# Resample source to match target
resampled <- resample(src_vol, targ_vol, interpolation=1)
```

`resampled_searchlight` *Create a resampled searchlight iterator*

Description

This function generates a resampled searchlight iterator by sampling regions from within a brain mask. By default it builds spherical searchlights, but users can provide a custom `shape_fun` to return ellipsoids, cubes, or arbitrary irregular searchlight shapes. Centers are drawn with replacement, so the same voxel (and its neighborhood) may appear multiple times. Each searchlight can also draw its radius from a user-specified set of radii.

Usage

```
resampled_searchlight(
  mask,
  radius = 8,
  iter = 100,
  shape_fun = NULL,
  nonzero = TRUE
)

bootstrap_searchlight(mask, radius = 8, iter = 100)
```

Arguments

`mask`	A `NeuroVol` object representing the brain mask.
`radius`	A numeric scalar or vector specifying candidate radii (in voxel units) for the searchlight sphere. If a vector is supplied, a radius is sampled uniformly (with replacement) for each searchlight. All radii must be positive. Default is 8.
`iter`	An integer specifying the total number of searchlights to sample (with replacement). Default is 100.
`shape_fun`	Either `NULL` (default spherical kernel), a character keyword (`"sphere"`, `"ellipsoid"`, `"cube"`, `"blobby"`), or a custom function. Custom functions are called as `shape_fun(mask, center, radius, iter, nonzero)` and must return either a `ROIVolWindow` or an `n x 3` integer matrix of voxel coordinates. This enables anisotropic or irregular searchlights.

`nonzero`	Logical; if `TRUE` (default), the generated searchlight is intersected with the nonzero voxels of `mask`. Applies to both the default sphere and any `shape_fun` that returns coordinates.

Details

Searchlight centers are sampled with replacement, so the same center (and its surrounding voxels) can be selected multiple times. When multiple radii are provided, each searchlight independently samples one radius from the supplied values. Supplying `shape_fun` lets you draw non-spherical searchlights (e.g., ellipsoids, cubes, blobby deformations, or task-specific kernels). Built-in shortcuts are available via `shape_fun = "ellipsoid"`, `"cube"`, and `"blobby"`; `"sphere"` or `NULL` uses the default spherical kernel.

Value

A `deferred_list` object containing `ROIVolWindow` objects, each representing a sampled searchlight region drawn from within the mask.

Examples

```
# Load an example brain mask
mask <- read_vol(system.file("extdata", "global_mask_v4.nii", package="neuroim2"))

# Generate a resampled searchlight iterator with radii drawn from {4,6,8}
searchlights <- resampled_searchlight(mask, radius = c(4, 6, 8))

# Use a custom shape: random ellipsoid scaled along each axis
ellipsoid_fun <- function(mask, center, radius, iter, nonzero) {
  scales <- runif(3, 0.5, 1.5)        # axis-wise stretch/compress
  vox <- spherical_roi(mask, center, radius, nonzero = FALSE)@coords
  ctr_mat <- matrix(center, nrow(vox), 3, byrow = TRUE)
  keep <- rowSums(((vox - ctr_mat) * scales)^2) <= radius^2
  vox[keep, , drop = FALSE]
}
ellip_searchlights <- resampled_searchlight(mask, radius = c(4, 6),
                                            iter = 50, shape_fun = ellipsoid_fun)

# Or use built-in named shapes
ellip_builtin <- resampled_searchlight(mask, radius = 6, shape_fun = "ellipsoid")
cube_builtin  <- resampled_searchlight(mask, radius = 6, shape_fun = "cube")
```

`resample_to` *Resample an image with readable method names*

Description

A convenience front-end to [resample()] that accepts human-friendly method names and an engine switch. Internally delegates to the S4 ‘resample(source, target, interpolation = 0/1/3)‘ methods.

Usage

```
resample_to(
  source,
  target,
  method = c("nearest", "linear", "cubic"),
  engine = c("internal"),
  ...
)
```

Arguments

`source`	A ‘NeuroVol‘ (source image)
`target`	A ‘NeuroVol‘ or ‘NeuroSpace‘ to match
`method`	Interpolation method: ‘"nearest"‘, ‘"linear"‘, or ‘"cubic"‘
`engine`	Resampling engine. For now only ‘"internal"‘ is supported.
`...`	Reserved for future options

Value

A ‘NeuroVol‘ in the target space

Examples

```
img <- read_vol(system.file("extdata","global_mask_v4.nii", package="neuroim2"))
sp  <- space(img)
sp2 <- NeuroSpace(sp@dim*2, sp@spacing/2, origin=sp@origin, trans=trans(img))
r1  <- resample_to(img, sp2, method = "linear")
```

`resolve_cmap` *Neuroimaging color palettes and helpers*

Description

Lightweight, perceptually-uniform color tools with safe fallbacks.

Usage

```
resolve_cmap(name = "grays", n = 256)
```

Arguments

`name`	Palette name (e.g., "grays", "viridis", "inferno", "magma", "plasma", "turbo", "cividis"). Case-insensitive. If you pass a vector of colors, it’s returned unchanged.
`n`	Number of colors to generate.

Value

A character vector of hex colors.

`ROI-class` *ROI*

Description

Base marker class for a region of interest (ROI)

`ROICoords` *Create ROI Coordinates Object*

Description

Creates an `ROICoords` object from a matrix of coordinates representing points in 3D space.

Usage

```
ROICoords(coords)
```

Arguments

`coords`	A matrix with 3 columns representing (x, y, z) coordinates

Details

ROI Coordinates

Value

An `ROICoords` object

Examples

```
coords <- matrix(c(1,2,3, 4,5,6), ncol=3, byrow=TRUE)
roi_coords <- ROICoords(coords)
```

`ROICoords-class` *ROICoords*

Description

A class representing a region of interest (ROI) in a brain image, defined by a set of coordinates. This class stores the geometric space of the image and the coordinates of the voxels within the ROI.

Slots

`space` An instance of class `NeuroSpace` representing the geometric space of the image data.

`coords` A `matrix` containing the coordinates of the voxels within the ROI. Each row represents a coordinate as, e.g. (i, j, k).

`ROIVec` *Create an instance of class* `ROIVec`

Description

This function constructs an instance of the ROIVec class, which represents a region of interest (ROI) in a 4D volume. The class stores the NeuroSpace object, voxel coordinates, and data values for the ROI.

Usage

```
ROIVec(vspace, coords, data = matrix(1, nrow = 1, ncol = nrow(coords)))
```

Arguments

`vspace`	An instance of class `NeuroSpace` with four dimensions, which represents the dimensions, voxel spacing, and time points of the 4D volume.
`coords`	A 3-column matrix of voxel coordinates for the region of interest.
`data`	The matrix of data values associated with the region of interest, with each row representing a voxel and each column representing a time point. By default, it is a matrix with a number of rows equal to the number of rows in the ‘coords‘ matrix and a single column filled with ones.

Value

An instance of class `ROIVec`, containing the NeuroSpace object, voxel coordinates, and data values for the region of interest.

Examples

```
# Create a NeuroSpace object
vspace <- NeuroSpace(dim = c(5, 5, 5, 10), spacing = c(1, 1, 1))

# Define voxel coordinates for the ROI
coords <- matrix(c(1, 2, 3, 2, 2, 2, 3, 3, 3), ncol = 3)

# Create a data matrix for the ROI
data <- matrix(rnorm(30), nrow = 10, ncol = 3)

# Create a ROIVec object
roi_vec <- ROIVec(vspace, coords, data)
```

`ROIVec-class` *ROIVec*

Description

A class representing a vector-valued volumetric region of interest (ROI) in a brain image.

Slots

- `coords` A `matrix` containing the 3D coordinates of the voxels within the ROI. Each row represents a voxel coordinate as (x, y, z).
- `.Data` A `matrix` containing the data values associated with each voxel in the ROI. Each row corresponds to a unique vector value, and the number of rows should match the number of rows in the `coords` matrix.

Validity

An object of class `ROIVec` is considered valid if: - The `coords` slot is a matrix with 3 columns. - The `.Data` slot is a matrix. - The number of rows in the `.Data` matrix is equal to the number of rows in the `coords` matrix.

`ROIVecWindow-class` *ROIVecWindow*

Description

A class representing a spatially windowed, vector-valued volumetric region of interest (ROI) in a brain image.

Slots

`coords` A `matrix` containing the 3D coordinates of the voxels within the ROI. Each row represents a voxel coordinate as (x, y, z).

`.Data` A `matrix` containing the data values associated with each voxel in the ROI. Each row corresponds to a unique vector value, and the number of rows should match the number of rows in the `coords` matrix.

`parent_index` An `integer` representing the 1D index of the center voxel in the parent space.

`center_index` An `integer` representing the location in the coordinate matrix of the center voxel in the window.

Validity

An object of class `ROIVecWindow` is considered valid if: `-` The `coords` slot is a matrix with 3 columns. `-` The `.Data` slot is a matrix. `-` The number of rows in the `.Data` matrix is equal to the number of rows in the `coords` matrix.

`ROIVol` *Create ROI Volume Object*

Description

Creates an `ROIVol` object representing a set of values at specific 3D coordinates within a spatial reference system.

Usage

```
ROIVol(space, coords, data)
```

Arguments

`space`	A `NeuroSpace` object defining the spatial reference
`coords`	A matrix with 3 columns representing (x,y,z) coordinates
`data`	A numeric vector of values corresponding to each coordinate

Details

ROI Volume

Value

An `ROIVol` object

Examples

```
space <- NeuroSpace(c(64,64,64))
coords <- matrix(c(1,2,3, 4,5,6), ncol=3, byrow=TRUE)
data <- c(1.5, 2.5)
roi_vol <- ROIVol(space, coords, data)
```

`ROIVol-class` *ROIVol*

Description

A class representing a volumetric region of interest (ROI) in a brain image, defined by a set of coordinates and associated data values.

Slots

`coords` A `matrix` containing the 3D coordinates of the voxels within the ROI. Each row represents a voxel coordinate as (x, y, z).

`.Data` A `numeric` vector containing the data values associated with each voxel in the ROI. The length of this vector should match the number of rows in the `coords` matrix.

Validity

An object of class `ROIVol` is considered valid if: - The `coords` slot is a matrix with 3 columns. - The `.Data` slot is a numeric vector. - The length of the `.Data` vector is equal to the number of rows in the `coords` matrix.

`ROIVolWindow-class` *ROIVolWindow*

Description

A class representing a spatially windowed volumetric region of interest (ROI) in a brain image, derived from a larger parent ROI.

Slots

`parent_index` An `integer` representing the 1D index of the center voxel in the parent space.

`center_index` An `integer` representing the location in the coordinate matrix of the center voxel in the window.

`coords` A `matrix` containing the 3D coordinates of the voxels within the ROI. Each row represents a voxel coordinate as (x, y, z).

`.Data` A `numeric` vector containing the data values associated with each voxel in the ROI. The length of this vector should match the number of rows in the `coords` matrix.

Validity

An object of class `ROIVolWindow` is considered valid if: - The `coords` slot is a matrix with 3 columns. - The `.Data` slot is a numeric vector. - The length of the `.Data` vector is equal to the number of rows in the `coords` matrix.

`scale` *Generic Scale Method*

Description

Scales an object by (typically) subtracting the mean and dividing by the standard deviation.

Usage

```
scale(x, ...)
```

Arguments

`x`	The object to be scaled.
`...`	Additional arguments for scaling methods.

Value

An object of the same class as `x`, scaled by the specified method.

`scale_fill_neuro` *A ggplot2 fill scale with neuroimaging-friendly defaults*

Description

A ggplot2 fill scale with neuroimaging-friendly defaults

Usage

```
scale_fill_neuro(
  cmap = "grays",
  range = c("robust", "data"),
  probs = c(0.02, 0.98),
  limits = NULL,
  na.value = "transparent",
  guide = "colorbar"
)
```

Arguments

`cmap`	Palette name or vector of colors. See [resolve_cmap()].
`range`	Either "robust" (quantiles) or "data" (min/max) to determine the default scale limits when ‘limits‘ is not provided.
`probs`	Two-length numeric vector of quantiles for ‘range="robust"‘.
`limits`	Optional numeric limits (min, max). Overrides ‘range‘.
`na.value`	Color for NA.
`guide`	Legend guide (default "colorbar").

Value

A ggplot2 scale object.

`scale_series` *Generic functions to scale (center and/or normalize by standard deviation) each series of a 4D image That is, if the 4th dimension is ’time’ each series is a 1D time series.*

Description

Generic functions to scale (center and/or normalize by standard deviation) each series of a 4D image That is, if the 4th dimension is ’time’ each series is a 1D time series.

Usage

```
scale_series(x, center, scale)

## S4 method for signature 'NeuroVec,logical,missing'
scale_series(x, center, scale)

## S4 method for signature 'NeuroVec,logical,logical'
scale_series(x, center, scale)

## S4 method for signature 'NeuroVec,missing,logical'
scale_series(x, center, scale)

## S4 method for signature 'NeuroVec,missing,missing'
scale_series(x, center, scale)
```

Arguments

`x`	a four dimensional image
`center`	a `logical` value indicating whether series should be centered
`scale`	a `logical` value indicating whether series should be divided by standard deviation

Value

An object of the same class as `x`, with each time series centered and/or scaled.

Examples

```
bvec <- NeuroVec(array(rnorm(24*24*24*24), c(24,24,24,24)), NeuroSpace(c(24,24,24,24), c(1,1,1)))
res <- scale_series(bvec, TRUE, TRUE)
```

`searchlight` *Create an exhaustive searchlight iterator*

Description

This function generates an exhaustive searchlight iterator that returns either voxel coordinates or ROIVolWindow objects for each searchlight sphere within the provided mask. The iterator visits every non-zero voxel in the mask as a potential center voxel.

Usage

```
searchlight(mask, radius, eager = FALSE, nonzero = FALSE, cores = 0)
```

Arguments

`mask`	A `NeuroVol` object representing the brain mask.
`radius`	A numeric value specifying the radius (in mm) of the spherical searchlight.
`eager`	A logical value specifying whether to eagerly compute the searchlight ROIs. Default is FALSE, which uses lazy evaluation.
`nonzero`	A logical value indicating whether to include only coordinates with nonzero values in the supplied mask. Default is FALSE.
`cores`	An integer specifying the number of cores to use for parallel computation. Default is 0, which uses a single core.

Value

A `deferred_list` object containing either matrices of integer-valued voxel coordinates or `ROIVolWindow` objects, each representing a searchlight region.

Examples

```
# Load an example brain mask
mask <- read_vol(system.file("extdata", "global_mask_v4.nii", package="neuroim2"))

# Generate an exhaustive searchlight iterator with a radius of 6 mm

searchlights <- searchlight(mask, radius = 6, eager = FALSE)
```

`searchlight-methods` *Searchlight Analysis Methods*

Description

Methods for performing searchlight analyses on neuroimaging data

---

`searchlight_coords` *Create an exhaustive searchlight iterator for voxel coordinates using spherical_roi*

Description

This function generates an exhaustive searchlight iterator that returns voxel coordinates for each searchlight sphere within the provided mask, using ‘spherical_roi‘ for neighborhood computation. The iterator visits every non-zero voxel in the mask as a potential center voxel.

Usage

```
searchlight_coords(mask, radius, nonzero = FALSE, cores = 0)
```

Arguments

`mask`	A `NeuroVol` object representing the brain mask.
`radius`	A numeric value specifying the radius (in mm) of the spherical searchlight.
`nonzero`	A logical value indicating whether to include only coordinates with nonzero values in the supplied mask. Default is FALSE.
`cores`	An integer specifying the number of cores to use for parallel computation. Default is 0, which uses a single core.

Value

A `deferred_list` object containing matrices of integer-valued voxel coordinates, each representing a searchlight region.

Examples

```
# Load an example brain mask
mask <- read_vol(system.file("extdata", "global_mask_v4.nii", package="neuroim2"))

# Generate an exhaustive searchlight iterator with a radius of 6 mm

searchlights <- searchlight_coords(mask, radius = 6)
```

`searchlight_shape_functions` *Convenience shape generators for* `resampled_searchlight()`

Description

Helpers that return ready-to-use `shape_fun` callbacks for `resampled_searchlight()`, covering a few sensible non-spherical defaults.

Usage

```
ellipsoid_shape(scales = c(1, 1, 1), jitter = 0)

cube_shape()

blobby_shape(drop = 0.3, edge_fraction = 0.7)
```

Arguments

`scales`	Length-3 positive numeric vector scaling the x/y/z axes relative to a sphere (for `ellipsoid_shape`). Values >1 stretch; <1 compress.
`jitter`	Non-negative numeric; standard deviation of multiplicative Gaussian noise applied to `scales` each draw (ellipsoid).
`drop`	Numeric in [0,1]; probability of dropping a voxel (blobby).
`edge_fraction`	Numeric in (0,1]; fraction of farthest voxels (by Euclidean distance from the center, in voxel units) considered "edge" and eligible for random dropping (blobby).

Details

Each returned function has signature `function(mask, center, radius, iter, nonzero)` and should return an $n \times 3$ integer coordinate matrix. The coordinates are later converted to a `ROIVolWindow` internally.

Value

A function suitable for the `shape_fun` argument of `resampled_searchlight()`.

Examples

```
mask <- read_vol(system.file("extdata", "global_mask_v4.nii", package="neuroim2"))

# Ellipsoid stretched along z with modest per-iteration jitter
sl_ellip <- resampled_searchlight(mask, radius = 6,
                                   shape_fun = ellipsoid_shape(scales = c(1, 1, 1.4),
                                                               jitter = 0.1))

# Simple axis-aligned cube (Chebyshev ball)
```

```
sl_cube <- resampled_searchlight(mask, radius = 5, shape_fun = "cube")

# Blobby sphere with 40% dropout on boundary voxels
sl_blob <- resampled_searchlight(mask, radius = 6,
                                 shape_fun = blobby_shape(drop = 0.4, edge_fraction = 0.6))
```

`series` *Extract one or more series from object*

Description

This function extracts time series data from specific voxel coordinates in a 4D neuroimaging object. It supports multiple ways of specifying the coordinates:

- Linear indices (1D)
- Grid coordinates (3D matrix)
- Individual x,y,z coordinates

Usage

```
series(x, i, ...)

## S4 method for signature 'ClusteredNeuroVec,numeric'
series(x, i, j, k, ...)

## S4 method for signature 'NeuroVec,matrix'
series(x, i)

## S4 method for signature 'NeuroVec,matrix'
series_roi(x, i)

## S4 method for signature 'NeuroVec,ROICoords'
series(x, i)

## S4 method for signature 'NeuroVec,ROICoords'
series_roi(x, i)

## S4 method for signature 'NeuroVec,LogicalNeuroVol'
series(x, i)

## S4 method for signature 'NeuroVec,NeuroVol'
series(x, i)

## S4 method for signature 'NeuroVec,LogicalNeuroVol'
series_roi(x, i)
```

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

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

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

## S4 method for signature 'NeuroVec,numeric'
series_roi(x, i, j, k, drop = TRUE)

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

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

## S4 method for signature 'NeuroVecSeq,matrix'
series(x, i)

## S4 method for signature 'NeuroVecSeq,matrix'
series_roi(x, i)

## S4 method for signature 'AbstractSparseNeuroVec,ROICoords'
series(x, i)

## S4 method for signature 'AbstractSparseNeuroVec,matrix'
series(x, i)

## S4 method for signature 'AbstractSparseNeuroVec,numeric'
series(x, i, j, k)

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

Arguments

`x`	A NeuroVecSeq object
`i`	A matrix of ROI coordinates (n x 3)
`...`	additional arguments
`j`	second dimension index
`k`	third dimension index
`drop`	whether to drop dimension of length 1

Value

A `list` or `array` containing the extracted series.

A matrix where each column represents a voxel's time series

A ROIVec object containing the time series for the specified ROI

See Also

series_roi

Examples

```
# Create a simple 4D neuroimaging vector (10x10x10 volume with 20 timepoints)
space <- NeuroSpace(c(10,10,10,20), c(1,1,1))
vec <- NeuroVec(array(rnorm(10*10*10*20), c(10,10,10,20)), space)

# Extract time series using linear indices
ts1 <- series(vec, 1:10)  # Get time series for first 10 voxels

# Extract time series using 3D coordinates
coords <- matrix(c(1,1,1, 2,2,2, 3,3,3), ncol=3, byrow=TRUE)
ts2 <- series(vec, coords)  # Get time series for 3 specific voxel locations

# Extract single time series using x,y,z coordinates
ts3 <- series(vec, 5, 5, 5)  # Get time series from middle voxel
```

---

series,NeuroHyperVec,ANY-method

Series method for NeuroHyperVec

Description

Series method for NeuroHyperVec

Usage

```
## S4 method for signature 'NeuroHyperVec,ANY'
series(x, i, j, k, ...)
```

Arguments

`x`	The NeuroHyperVec object
`i`	first index
`j`	second index
`k`	third index
`...`	Additional arguments (not used)

Details

when x is a NeuroHyperVec object, the series method returns a 2D array with dimensions [features x trials]

Value

A 2D array with dimensions [features x trials]

`series_roi`	*Extract time series from specific voxel coordinates and return as ROI object*

Description

Extracts time series data from a `NeuroVec` object at specified voxel coordinates and returns it as an ROI object.

Usage

```
series_roi(x, i, ...)
```

Arguments

`x`	The `NeuroVec` object
`i`	Numeric index for the first dimension
`...`	Additional arguments

Value

A `ROIVec` object containing the time series data for the specified coordinates.

See Also

`series`

Examples

```
# Create a simple 4D neuroimaging vector
space <- NeuroSpace(c(10,10,10,20), c(1,1,1))
vec <- NeuroVec(array(rnorm(10*10*10*20), c(10,10,10,20)), space)

# Extract time series for first 100 voxels as ROI
roi1 <- series_roi(vec, 1:100)

# Extract time series using 3D coordinates
coords <- matrix(c(1,1,1, 2,2,2, 3,3,3), ncol=3, byrow=TRUE)
roi2 <- series_roi(vec, coords)
```

`show,NamedAxis-method` *Show method for NamedAxis objects*

Description

Show method for NamedAxis objects

Show method for ClusteredNeuroVec

Usage

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

## S4 method for signature 'AxisSet1D'
show(object)

## S4 method for signature 'AxisSet2D'
show(object)

## S4 method for signature 'AxisSet3D'
show(object)

## S4 method for signature 'AxisSet4D'
show(object)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Arguments

`object`	A NamedAxis object

Value

Invisibly returns `NULL`, called for its side effect of displaying the object.

---

simulate_fmri *Simulate fMRI Data*

Description

Generates synthetic 4D fMRI data with realistic spatiotemporal properties including temporal autocorrelation, spatial smoothness, heteroscedasticity, and optional global signal fluctuations and latent components.

Usage

```
simulate_fmri(
  mask,
  n_time,
  TR = 2,
  spatial_fwhm = 6,
  ar_mean = 0.45,
  ar_sd = 0.08,
  noise_sd = 1,
  hetero_fwhm = 20,
  hetero_strength = 0.6,
  global_amp = 0.2,
  global_rho = 0.85,
  n_factors = 4,
  factor_fwhm = 12,
  factor_rho = 0.8,
  seed = NULL,
  return_centered = TRUE
)
```

Arguments

`mask`	A `NeuroVol` object defining the brain mask region. Can be binary or continuous (non-zero values define the mask).
`n_time`	Integer specifying the number of time points to simulate.
`TR`	Numeric value for the repetition time in seconds (default = 2.0). Currently used only for metadata.
`spatial_fwhm`	Numeric value specifying the spatial smoothness in mm (full width at half maximum) applied to each timepoint (default = 6).
`ar_mean`	Numeric value for the mean of the AR(1) coefficient distribution across voxels (default = 0.45).
`ar_sd`	Numeric value for the standard deviation of the AR(1) coefficient distribution (default = 0.08).
`noise_sd`	Numeric value for the baseline noise standard deviation (default = 1.0).

`hetero_fwhm`	Numeric value for the spatial scale (FWHM in mm) of the heteroscedasticity field (default = 20).
`hetero_strength`	Numeric value controlling the strength of spatial heteroscedasticity on log scale (default = 0.6).
`global_amp`	Numeric value for the amplitude of global signal fluctuations as a fraction of median noise (default = 0.2). Set to 0 to disable.
`global_rho`	Numeric value for the AR(1) coefficient of global signal (default = 0.85).
`n_factors`	Integer specifying the number of latent spatial components (default = 4). Set to 0 to disable.
`factor_fwhm`	Numeric value for the spatial smoothness (FWHM in mm) of latent component maps (default = 12).
`factor_rho`	Numeric value for the AR(1) coefficient of latent component time courses (default = 0.8).
`seed`	Integer seed for random number generation (default = NULL for no seed).
`return_centered`	Logical indicating whether to center each voxel's time series to mean zero (default = TRUE).

Details

The simulation combines several realistic features:

- Voxel-wise AR(1) temporal autocorrelation with spatial variation
- Spatial smoothing applied to innovations for realistic spatial correlation
- Heteroscedastic noise with smooth spatial modulation
- Optional low-frequency global signal fluctuations
- Optional latent spatial components resembling resting-state networks

The spatial smoothing uses the package's optimized `gaussian_blur` function for efficiency.

Value

A `NeuroVec` object containing the simulated 4D fMRI data.

References

Welvaert, M., & Rosseel, Y. (2013). On the definition of signal-to-noise ratio and contrast-to-noise ratio for fMRI data. PloS one, 8(11), e77089.

Examples

```
# Create a simple spherical mask
dims <- c(32, 32, 20)
mask_array <- array(FALSE, dims)
center <- dims / 2
for (i in 1:dims[1]) {
```

```
    for (j in 1:dims[2]) {
      for (k in 1:dims[3]) {
        if (sum(((c(i,j,k) - center) / (dims/3))^2) <= 1) {
          mask_array[i,j,k] <- TRUE
        }
      }
    }
  }

  mask <- NeuroVol(mask_array, NeuroSpace(dims, c(3,3,3)))

  # Simulate 100 time points
  sim_data <- simulate_fmri(mask, n_time = 100, seed = 42)

  # Check dimensions
  dim(sim_data)  # Should be c(32, 32, 20, 100)
```

`slice` *Extract image slice*

Description

Extract a 2D slice from an image volume

Usage

```
slice(x, zlevel, along, orientation, ...)

## S4 method for signature 'NeuroVol,numeric,numeric,missing'
slice(x, zlevel, along, orientation)

## S4 method for signature 'NeuroVol,numeric,NeuroSpace,AxisSet3D'
slice(x, zlevel, along, orientation)
```

Arguments

`x`	the object
`zlevel`	coordinate (in voxel units) along the sliced axis
`along`	the axis along which to slice
`orientation`	the target orientation of the 2D slice
`...`	additional arguments

Value

A 2D slice from the image volume.

`slices` *Extract an ordered series of 2D slices from a 3D or 4D object*

Description

This function extracts an ordered series of 2D slices from a 3D or 4D object. The returned slices are in the order they appear in the original object.

Usage

```
slices(x, ...)

## S4 method for signature 'NeuroVol'
slices(x)
```

Arguments

`x`	A NeuroVol object
`...`	Additional arguments to be passed to the underlying methods

Value

A `list` of 2D `matrices`, each containing a slice from the input `x`.

A deflist object containing functions that return 2D slices of the volume along the z-axis. The length of the deflist equals the number of slices in the z dimension.

Examples

```
# Create a simple 3D volume
space <- NeuroSpace(c(10,10,10), c(1,1,1))
vol <- NeuroVol(array(rnorm(10*10*10), c(10,10,10)), space)

# Get all slices along the z-axis
slc <- slices(vol)

# Number of slices equals the z dimension
length(slc) == dim(vol)[3]

# Each slice is a 2D matrix
dim(slc[[1]]) == c(10,10)
```

space *Extract Geometric Properties of an Image*

Description

This function retrieves the geometric properties of a given image, such as dimensions and voxel size.

Retrieves the NeuroSpace object associated with an IndexLookupVol object.

Usage

```
space(x, ...)

## S4 method for signature 'ClusteredNeuroVec'
space(x)

## S4 method for signature 'IndexLookupVol'
space(x)

## S4 method for signature 'ROICoords'
space(x)

## S4 method for signature 'NeuroObj'
space(x)

## S4 method for signature 'NeuroSpace'
space(x)
```

Arguments

`x`	An `IndexLookupVol` object
`...`	Additional arguments, if needed.

Value

A `NeuroSpace` object representing the geometric space of `x`.

Examples

```
# Create a NeuroSpace object with dimensions (10, 10, 10) and voxel size (1, 1, 1)
x <- NeuroSpace(c(10, 10, 10), c(1, 1, 1))

# Create a NeuroVol object with random data and the specified NeuroSpace
vol <- NeuroVol(rnorm(10 * 10 * 10), x)

# Retrieve the geometric properties of the NeuroVol object
identical(x, space(vol))
```

```
space <- NeuroSpace(c(64, 64, 64), c(1, 1, 1), c(0, 0, 0))
ilv <- IndexLookupVol(space, c(1:100))
space(ilv)  # Get the associated NeuroSpace object
```

`spacing` *Extract Voxel Dimensions of an Image*

Description

This function extracts the voxel dimensions of an image represented by the input object.

Usage

```
spacing(x)

## S4 method for signature 'ROICoords'
spacing(x)

## S4 method for signature 'NeuroObj'
spacing(x)

## S4 method for signature 'NeuroSpace'
spacing(x)
```

Arguments

`x`	The object representing the image.

Value

A numeric vector specifying the voxel dimensions of `x`.

Examples

```
bspace <- NeuroSpace(c(10, 10, 10), c(2, 2, 2))
all.equal(spacing(bspace), c(2, 2, 2))
```

`SparseNeuroVec-class` *SparseNeuroVec Class*

Description

A class representing a sparse four-dimensional brain image, optimized for efficient storage and access of large, sparse neuroimaging data.

Constructs a SparseNeuroVec object for efficient representation and manipulation of sparse neuroimaging data with many zero or missing values.

Usage

```
SparseNeuroVec(data, space, mask, label = "")
```

Arguments

`data`	A matrix or a 4-D array containing the neuroimaging data. The dimensions of the data should be consistent with the dimensions of the provided NeuroSpace object and mask.
`space`	A NeuroSpace object representing the dimensions and voxel spacing of the neuroimaging data.
`mask`	A 3D array, 1D vector of type logical, or an instance of type LogicalNeuroVol, which specifies the locations of the non-zero values in the data.
`label`	Optional character string providing a label for the vector

Details

SparseNeuroVec objects store data in a compressed format, where only non-zero values are retained. This approach significantly reduces memory usage for sparse brain images. The class leverages the mask and mapping from its parent class `AbstractSparseNeuroVec` to efficiently manage the spatial structure of the data.

Value

A SparseNeuroVec object, containing the sparse neuroimaging data, mask, and associated NeuroSpace information.

Slots

`data` A `matrix` where each column represents a non-zero vector spanning the fourth dimension (e.g., time series for each voxel). Rows correspond to voxels in the sparse domain defined by the mask.

Inheritance

`SparseNeuroVec` inherits from:

- `NeuroVec`: Base class for 4D brain images
- `AbstractSparseNeuroVec`: Provides sparse representation framework
- `ArrayLike4D`: Interface for 4D array-like operations

See Also

`AbstractSparseNeuroVec-class` for the parent sparse representation class. `NeuroVec-class` for the base 4D brain image class.

Examples

```
# Create a sparse 4D brain image
mask <- LogicalNeuroVol(array(runif(64*64*32) > 0.7, c(64,64,32)), NeuroSpace(c(64,64,32)))
data <- matrix(rnorm(sum(mask) * 100), nrow=sum(mask), ncol=100)
sparse_vec <- SparseNeuroVec(data=data, mask=mask, space=NeuroSpace(dim=c(64,64,32,100)))

# Access a subset of the data
subset <- sparse_vec[,,, 1:10]


bspace <- NeuroSpace(c(10,10,10,100), c(1,1,1))
mask <- array(rnorm(10*10*10) > .5, c(10,10,10))
mat <- matrix(rnorm(sum(mask)), 100, sum(mask))
svec <- SparseNeuroVec(mat, bspace, mask)
length(indices(svec)) == sum(mask)
```

`SparseNeuroVecSource-class`

SparseNeuroVecSource Class

Description

A class used to produce a `SparseNeuroVec` instance. It encapsulates the necessary information to create a sparse representation of a 4D neuroimaging dataset.

Details

SparseNeuroVecSource acts as a factory for SparseNeuroVec objects. It holds the spatial mask that determines which voxels will be included in the sparse representation. This class is typically used in data loading or preprocessing pipelines where the sparse structure of the data is known or determined before the full dataset is loaded.

Slots

`mask` An object of class `LogicalNeuroVol` representing the subset of voxels that will be stored in memory. This mask defines the sparse structure of the resulting SparseNeuroVec.

Inheritance

`SparseNeuroVecSource` inherits from:

- `NeuroVecSource`: Base class for NeuroVec source objects

See Also

`SparseNeuroVec-class` for the resulting sparse 4D neuroimaging data class. `LogicalNeuroVol-class` for the mask representation.

Examples

```
# Create a simple mask
mask_data <- array(runif(64*64*32) > 0.7, dim = c(64, 64, 32))
mask <- LogicalNeuroVol(mask_data, space = NeuroSpace(dim = c(64, 64, 32)))

# Create a SparseNeuroVecSource
sparse_source <- new("SparseNeuroVecSource", mask = mask)
```

`SparseNeuroVol-class` *SparseNeuroVol Class*

Description

This class represents a three-dimensional brain image using a sparse data representation. It is particularly useful for large brain images with a high proportion of zero or missing values, offering efficient storage and processing.

Construct a `SparseNeuroVol` instance

Usage

```
SparseNeuroVol(data, space, indices = NULL, label = "")
```

Arguments

`data`	a numeric vector or ROIVol
`space`	an instance of class `NeuroSpace`
`indices`	a index vector indicating the 1-d coordinates of the data values
`label`	a `character` string

Details

The SparseNeuroVol class extends the `NeuroVol` class and implements the ArrayLike3D interface. It uses a `sparseVector` from the Matrix package to store the image data, which allows for memory-efficient representation of sparse 3D neuroimaging data.

Image data is backed by `Matrix::sparseVector`.

Value

`SparseNeuroVol` instance

Slots

`data` A `sparseVector` object from the Matrix package, storing the image volume data in a sparse format.

References

Bates, D., & Maechler, M. (2019). Matrix: Sparse and Dense Matrix Classes and Methods. R package version 1.2-18. https://CRAN.R-project.org/package=Matrix

See Also

`NeuroVol-class` for the base volumetric image class. `DenseNeuroVol-class` for a dense representation of 3D brain images.

Examples

```
# Create a sparse 3D brain image
dim <- c(64L, 64L, 64L)
space <- NeuroSpace(dim = dim, origin = c(0, 0, 0), spacing = c(1, 1, 1))
sparse_data <- Matrix::sparseVector(x = c(1, 2, 3),
                                    i = c(100, 1000, 10000),
                                    length = prod(dim))
sparse_vol <- new("SparseNeuroVol", space = space, data = sparse_data)
sparse_vol[1000] == 1

data <- 1:10
indices <- seq(1,1000, length.out=10)
bspace <- NeuroSpace(c(64,64,64), spacing=c(1,1,1))
sparsevol <- SparseNeuroVol(data,bspace,indices=indices)
densevol <- NeuroVol(data,bspace,indices=indices)
sum(sparsevol) == sum(densevol)
```

`spatial-filter` *Spatial Filtering Methods for Neuroimaging Data*

Description

Methods for applying spatial filters to neuroimaging data

---

spherical_roi *Create a Spherical Region of Interest*

Description

Creates a Spherical ROI based on a centroid.

Usage

```
spherical_roi(
  bvol,
  centroid,
  radius,
  fill = NULL,
  nonzero = FALSE,
  use_cpp = TRUE
)
```

Arguments

`bvol`	an `NeuroVol` or `NeuroSpace` instance
`centroid`	the center of the sphere in positive-coordinate (i,j,k) voxel space.
`radius`	the radius in real units (e.g. millimeters) of the spherical ROI
`fill`	optional value(s) to store as data
`nonzero`	if `TRUE`, keep only nonzero elements from `bvol`
`use_cpp`	whether to use compiled c++ code

Value

an instance of class `ROIVol`

See Also

[spherical_roi_set()] for efficiently creating many spherical ROIs, [series_roi()] and [coords()] for extracting time series and coordinates from ROIs, and the vignette: `vignette("regionOfInterest", package = "neuroim2")`.

Examples

```
sp1 <- NeuroSpace(c(10,10,10), c(1,2,3))
# create an ROI centered around the integer-valued positive voxel coordinate: i=5, j=5, k=5
cube <- spherical_roi(sp1, c(5,5,5), 3.5)
vox <- coords(cube)
cds <- coords(cube, real=TRUE)
## fill in ROI with value of 6
cube1 <- spherical_roi(sp1, c(5,5,5), 3.5, fill=6)
all(cube1 == 6)
```

```
## Create multiple spherical ROIs at once (preferred):
centers <- rbind(c(5,5,5), c(3,3,3), c(7,7,7))
vols <- spherical_roi_set(bvol = sp1,
                         centroids = centers, radius = 3.5, fill = 1)
length(vols)  # 3

## Equivalent, less efficient lapply variant:
vols2 <- lapply(seq_len(nrow(centers)), function(i) {
  spherical_roi(sp1, centers[i,], radius = 3.5, fill = 1)
})

# create an ROI centered around the real-valued coordinates: x=5, y=5, z=5
vox <- coord_to_grid(sp1, c(5, 5, 5))
cube <- spherical_roi(sp1, vox, 3.5)
```

`spherical_roi_set` *Create Multiple Spherical Regions of Interest*

Description

This function generates multiple spherical ROIs simultaneously, centered at the provided voxel coordinates. It is more efficient than calling `spherical_roi` multiple times when you need to create many ROIs.

Usage

```
spherical_roi_set(bvol, centroids, radius, fill = NULL, nonzero = FALSE)
```

Arguments

`bvol`	A `NeuroVol` or `NeuroSpace` instance
`centroids`	A matrix of voxel coordinates where each row represents a centroid (i,j,k)
`radius`	The radius in real units (e.g. millimeters) of the spherical ROIs
`fill`	Optional value(s) to store as data. If provided, must be either a single value or a vector with length equal to the number of ROIs
`nonzero`	If `TRUE`, keep only nonzero elements from `bvol`

Value

A list of `ROIVolWindow` objects, one for each centroid

Examples

```
# Create a NeuroSpace object
sp1 <- NeuroSpace(c(10,10,10), c(1,2,3))

# Create multiple ROIs centered at different voxel coordinates
centroids <- matrix(c(5,5,5, 3,3,3, 7,7,7), ncol=3, byrow=TRUE)
rois <- spherical_roi_set(sp1, centroids, 3.5)

# Create ROIs with specific fill values
rois <- spherical_roi_set(sp1, centroids, 3.5, fill=c(1,2,3))
```

`split_blocks` *Cut a vector-valued object into a list of sub-blocks*

Description

Splits a vector-valued object into a list of sub-blocks defined by a vector of indices.

Usage

```
split_blocks(x, indices, ...)

## S4 method for signature 'NeuroVec,integer'
split_blocks(x, indices, ...)

## S4 method for signature 'NeuroVec,factor'
split_blocks(x, indices, ...)

## S4 method for signature 'NeuroVec,factor'
split_blocks(x, indices, ...)
```

Arguments

`x`	a vector-valued object
`indices`	a vector of indices defining the sub-blocks. Must match the length of the input vector.
`...`	additional arguments

Value

A `list` of sub-blocks, where each sub-block contains the elements from `x` corresponding to the matching `indices`.

Examples

```
# Create a 4D neuroimaging vector with 20 timepoints
space <- NeuroSpace(c(10,10,10,20), c(1,1,1))
vec <- NeuroVec(array(rnorm(10*10*10*20), c(10,10,10,20)), space)

# Split into 4 blocks by assigning timepoints to blocks 1-4 repeatedly
block_indices <- rep(1:4, length.out=20)
blocks <- split_blocks(vec, block_indices)
```

`split_clusters` *Cut an object into a list of spatial or spatiotemporal clusters*

Description

This function cuts an object into a list of sub-objects based on a vector of cluster indices. The resulting list contains each of the clusters as separate objects.

These methods split a NeuroVec object into multiple ROIVec objects based on cluster assignments.

Usage

```
split_clusters(x, clusters, ...)

## S4 method for signature 'NeuroVec,ClusteredNeuroVol'
split_clusters(x, clusters, ...)

## S4 method for signature 'NeuroVec,integer'
split_clusters(x, clusters, ...)

## S4 method for signature 'NeuroVol,ClusteredNeuroVol'
split_clusters(x, clusters)

## S4 method for signature 'NeuroVol,integer'
split_clusters(x, clusters)

## S4 method for signature 'NeuroVol,numeric'
split_clusters(x, clusters)

## S4 method for signature 'ClusteredNeuroVol,missing'
split_clusters(x, clusters)

## S4 method for signature 'NeuroVec,integer'
split_clusters(x, clusters, ...)

## S4 method for signature 'NeuroVec,numeric'
split_clusters(x, clusters, ...)
```

```
## S4 method for signature 'NeuroVec,ClusteredNeuroVol'
split_clusters(x, clusters, ...)
```

Arguments

`x`	A NeuroVec object to be split.
`clusters`	Either a ClusteredNeuroVol object or an integer vector of cluster assignments.
`...`	Additional arguments to be passed to methods.

Details

There are two methods for splitting clusters:

- Using a ClusteredNeuroVol object: This method uses the pre-defined clusters in the ClusteredNeuroVol object.
- Using an integer vector: This method allows for custom cluster assignments.

methods return a deflist, which is a lazy-loading list of ROIVec objects.

Value

A `list` of sub-objects, where each sub-object corresponds to a unique cluster index.

A deflist (lazy-loading list) of ROIVec objects, where each element corresponds to a cluster.

See Also

`NeuroVec-class`, `ClusteredNeuroVol-class`, `ROIVec-class`

Examples

```
# Create a synthetic 3D volume and its NeuroSpace
space <- NeuroSpace(c(10, 10, 10,4))
vol_data <- array(rnorm(10 * 10 * 10 * 4), dim = c(10, 10, 10,4))
neuro_vec <- NeuroVec(vol_data, space)

# Create a binary mask (e.g., select voxels with values > 0)
mask_data <- as.logical(neuro_vec[[1]] > .5)
mask_vol <- LogicalNeuroVol(mask_data, NeuroSpace(c(10, 10, 10)))

# Extract indices and coordinates for the masked voxels
mask_idx <- which(mask_data)
coords <- index_to_coord(mask_vol, mask_idx)

# Perform k-means clustering on the coordinates (e.g., 3 clusters)
set.seed(123)  # for reproducibility
k_res <- kmeans(coords, centers = 3)

# Create a ClusteredNeuroVol using the mask and k-means cluster assignments
clustered_vol <- ClusteredNeuroVol(mask_vol, k_res$cluster)

# Split the NeuroVec by clusters using the ClusteredNeuroVol method
```

```
  split_result_clust <- split_clusters(neuro_vec, clustered_vol)

  # Calculate and print the mean value for each cluster
  means_clust <- sapply(split_result_clust, function(x) mean(values(x)))
  print(means_clust)

  # Alternatively, create an integer vector of cluster assignments:
  cluster_assignments <- numeric(prod(dim(space)[1:3]))
  cluster_assignments[mask_idx] <- k_res$cluster
  split_result_int <- split_clusters(neuro_vec, as.integer(cluster_assignments))

  # Verify that both splitting methods yield the same cluster means
  means_int <- sapply(split_result_int, function(x) mean(values(x)))
  print(all.equal(sort(means_clust), sort(means_int)))



# Create a simple example space and data
space <- NeuroSpace(c(10, 10, 10,4))
data <- array(rnorm(1000*4), dim = c(10, 10, 10,4))
vec <- NeuroVec(data, space)

# Create a mask for clustering (e.g., values > 0)
mask <- vec[,,,1] > 0
mask_vol <- LogicalNeuroVol(as.array(mask), NeuroSpace(c(10, 10, 10)))

# Get coordinates of masked voxels for clustering
mask_idx <- which(mask)
coords <- index_to_coord(mask_vol, mask_idx)

# Perform clustering on the coordinates (3 clusters for example)
set.seed(123) # for reproducibility
kmeans_result <- kmeans(coords, centers = 3)

# Create a ClusteredNeuroVol
clustered_vol <- ClusteredNeuroVol(mask_vol, kmeans_result$cluster)

# Split the NeuroVec by clusters
split_result <- split_clusters(vec, clustered_vol)

# Calculate mean value for each cluster
cluster_means <- sapply(split_result, function(x) mean(values(x)))
print(cluster_means)

# Alternative: using integer cluster assignments
cluster_indices <- numeric(prod(dim(space)[1:3]))
cluster_indices[mask_idx] <- kmeans_result$cluster
split_result2 <- split_clusters(vec, as.integer(cluster_indices))

# Verify both methods give same results
cluster_means2 <- sapply(split_result2, function(x) mean(values(x)))
print(all.equal(sort(cluster_means), sort(cluster_means2)))
```

---

split_fill — *Fill Disjoint Sets of Values with the Output of a Function*

Description

This function splits an object into disjoint sets of values based on a factor, applies a specified function to each set, and returns a new object with the original values replaced by the function's output.

Usage

```
split_fill(x, fac, FUN)

## S4 method for signature 'NeuroVol,factor,function'
split_fill(x, fac, FUN)
```

Arguments

`x`	The object to split.
`fac`	The `factor` to split by.
`FUN`	The function used to summarize the sets.

Details

The `FUN` function can either return a scalar for each input vector or a vector equal to the length of the input vector. If it returns a scalar, every voxel in the set will be filled with that value in the output vector.

Value

An object of the same class as `x`, with values replaced by the output of `FUN`.

Examples

```
## Summarize with mean -- FUN returns a scalar
x <- NeuroSpace(c(10, 10, 10), c(1, 1, 1))
vol <- NeuroVol(rnorm(10 * 10 * 10), x)
fac <- factor(rep(1:10, length.out=1000))
ovol.mean <- split_fill(vol, fac, mean)
identical(dim(ovol.mean), dim(vol))
length(unique(as.vector(ovol.mean))) == 10

## Transform by reversing vector -- FUN returns a vector
ovol2 <- split_fill(vol, fac, rev)
```

`split_reduce` *Summarize Subsets of an Object by Splitting by Row and Applying a Summary Function*

Description

This function summarizes subsets of a numeric matrix or matrix-like object by first splitting the object by row and then applying a summary function.

Usage

```
split_reduce(x, fac, FUN)

## S4 method for signature 'matrix,integer,function'
split_reduce(x, fac, FUN)

## S4 method for signature 'matrix,factor,missing'
split_reduce(x, fac)

## S4 method for signature 'matrix,factor,function'
split_reduce(x, fac, FUN)

## S4 method for signature 'NeuroVec,factor,function'
split_reduce(x, fac, FUN)

## S4 method for signature 'NeuroVec,factor,missing'
split_reduce(x, fac, FUN)
```

Arguments

`x`	A numeric matrix or matrix-like object.
`fac`	A factor to define subsets of the object.
`FUN`	The summary function to apply to each subset. If not provided, the mean of each sub-matrix column is computed.

Details

If 'FUN' is supplied, it must take a vector and return a single scalar value. If it returns more than one value, an error will occur.

If 'x' is a NeuroVec instance, voxels (dimensions 1:3) are treated as columns and time-series (dimension 4) as rows. The summary function is then applied to groups of voxels. However, if the goal is to apply a function to groups of time-points.

Value

A `matrix` (or matrix-like object) containing the summarized values after applying `FUN`.

Examples

```
mat = matrix(rnorm(100*100), 100, 100)
fac = factor(sample(1:3, nrow(mat), replace=TRUE))
## Compute column means of each sub-matrix
ms <- split_reduce(mat, fac)
all.equal(row.names(ms), levels(fac))

## Compute column medians of each sub-matrix
ms <- split_reduce(mat, fac, median)

## Compute time-series means grouped over voxels.
## Here, 'length(fac)' must equal the number of voxels: 'prod(dim(bvec)[1:3])'
bvec <- NeuroVec(array(rnorm(24*24*24*24), c(24,24,24,24)), NeuroSpace(c(24,24,24,24), c(1,1,1)))
fac <- factor(sample(1:3, prod(dim(bvec)[1:3]), replace=TRUE))
ms <- split_reduce(bvec, fac)
ms2 <- split_reduce(bvec, fac, mean)
all.equal(row.names(ms), levels(fac))
all.equal(ms, ms2)
```

`split_scale` *Center and/or Scale Row-subsets of a Matrix or Matrix-like Object*

Description

This function centers and/or scales the row-subsets of a numeric matrix or matrix-like object.

Usage

```
split_scale(x, f, center, scale)

## S4 method for signature 'matrix,factor,logical,logical'
split_scale(x, f, center = TRUE, scale = TRUE)

## S4 method for signature 'matrix,factor,missing,missing'
split_scale(x, f)

## S4 method for signature 'DenseNeuroVec,factor,missing,missing'
split_scale(x, f)

## S4 method for signature 'DenseNeuroVec,factor,logical,missing'
split_scale(x, f, center)

## S4 method for signature 'DenseNeuroVec,factor,logical,logical'
split_scale(x, f, center, scale)
```

Arguments

`x`	A numeric matrix or matrix-like object.
`f`	The splitting object, typically a factor or a set of integer indices. Must be equal to the number of rows in the matrix.
`center`	Should values within each submatrix be centered? If TRUE, the mean is removed from each column of the submatrix.
`scale`	Should values be scaled? If TRUE, the vector is divided by the standard deviation for each column of the submatrix.

Value

An object of the same class as `x`, with row-subsets centered and/or scaled according to `f`.

Examples

```
M <- matrix(rnorm(1000), 10, 100)
fac <- factor(rep(1:2, each=5))
Ms <- split_scale(M, fac)

## Correctly centered
all(abs(apply(Ms[fac == 1,], 2, mean)) < .000001)
all(abs(apply(Ms[fac == 2,], 2, mean)) < .000001)

## Correctly scaled
all.equal(apply(Ms[fac == 1,], 2, sd), rep(1, ncol(Ms)))
all.equal(apply(Ms[fac == 2,], 2, sd), rep(1, ncol(Ms)))
```

`square_roi` *Create a square region of interest*

Description

This function creates a square region of interest (ROI) in a 3D volume, where the z-dimension is fixed at one voxel coordinate. The ROI is defined within a given NeuroVol or NeuroSpace instance.

Usage

```
square_roi(bvol, centroid, surround, fill = NULL, nonzero = FALSE, fixdim = 3)
```

Arguments

`bvol`	A `NeuroVol` or `NeuroSpace` instance representing the 3D volume or space.
`centroid`	A numeric vector of length 3, representing the center of the square ROI in voxel coordinates.
`surround`	A non-negative integer specifying the number of voxels on either side of the central voxel.

`fill`	An optional value or values to assign to the data slot of the resulting ROI. If not provided, no data will be assigned.
`nonzero`	A logical value indicating whether to keep only nonzero elements from `bvol`. If `bvol` is a `NeuroSpace` instance, this argument is ignored.
`fixdim`	A logical value indicating whether the fixed dimension is the third, or z, dimension. Default is TRUE.

Value

An instance of class `ROIVol` representing the square ROI.

Examples

```
sp1 <- NeuroSpace(c(10, 10, 10), c(1, 1, 1))
square <- square_roi(sp1, c(5, 5, 5), 1)
vox <- coords(square)
## a 3 X 3 X 1 grid
nrow(vox) == 9
```

`strip_extension` *Generic function to strip extension from file name, given a* `FileFormat` *instance.*

Description

Removes the file extension from a given file name based on the FileFormat specifications.

Usage

```
strip_extension(x, file_name)

## S4 method for signature 'FileFormat,character'
strip_extension(x, file_name)
```

Arguments

`x`	A FileFormat object specifying the format requirements
`file_name`	A character string specifying the file name to strip the extension from

Details

The function performs the following steps:

1. If the file_name matches the header file format, it removes the header extension.
2. If the file_name matches the data file format, it removes the data extension.
3. If the file_name doesn't match either format, it throws an error.

Value

A `character` string `file_name` without its extension.

A character string representing the file name without the extension

See Also

`header_file`, `data_file` for related file name manipulation

Examples

```
# Create a FileFormat for NIFTI files
fmt <- new("FileFormat",
           header_extension = "nii",
           data_extension = "nii")

# Strip extension from a NIFTI file
strip_extension(fmt, "brain_scan.nii")  # Returns "brain_scan"
```

`sub_vector` *Generic function to extract a sub-vector from a* `NeuroVec` *object.*

Description

Extracts a subset of volumes from a file-backed neuroimaging vector and returns them as a dense (in-memory) vector.

Extracts a subsequence of volumes from a NeuroVecSeq object.

Usage

```
sub_vector(x, i, ...)

## S4 method for signature 'FileBackedNeuroVec,numeric'
sub_vector(x, i)

## S4 method for signature 'NeuroVec,numeric'
sub_vector(x, i)

## S4 method for signature 'NeuroVecSeq,numeric'
sub_vector(x, i)

## S4 method for signature 'NeuroVecSeq,numeric'
sub_vector(x, i)

## S4 method for signature 'SparseNeuroVec,numeric'
sub_vector(x, i)
```

Arguments

`x`	A NeuroVecSeq object
`i`	Numeric vector of indices specifying the time points to extract
`...`	additional arguments

Details

This method efficiently reads only the requested volumes from disk, converting them to an in-memory representation. The spatial metadata is preserved but adjusted to reflect the new number of volumes.

Memory usage is proportional to the number of volumes requested, not the size of the full dataset.

Value

A `NeuroVec` object that is a sub-sequence of the supplied object.

A NeuroVecSeq object containing the extracted subsequence

Examples

```
bvec <- NeuroVec(array(rnorm(24*24*24*24), c(24,24,24,24)), NeuroSpace(c(24,24,24,24), c(1,1,1)))
vec <- sub_vector(bvec,1:2)
all.equal(2, dim(vec)[4])

vec <- sub_vector(bvec, c(1,3,5,7))
all.equal(4, dim(vec)[4])

mask <- LogicalNeuroVol(rep(TRUE, 24*24*24), NeuroSpace(c(24,24,24), c(1,1,1)))
svec <- SparseNeuroVec(array(rnorm(24*24*24*24), c(24,24,24,24)),
NeuroSpace(c(24,24,24,24), c(1,1,1)), mask)
vec <- sub_vector(svec, c(1,3,5))
all.equal(3, dim(vec)[4])
```

`Summary-methods` *Summary Methods for Neuroimaging Objects*

Description

Methods for the `Summary` group generic (e.g., `sum`, `min`, `max`, `range`, `prod`, `any`, `all`) applied to neuroimaging data objects.

Usage

```
## S4 method for signature 'SparseNeuroVec'
Summary(x, ..., na.rm = FALSE)

## S4 method for signature 'SparseNeuroVol'
Summary(x, ..., na.rm = FALSE)
```

```
## S4 method for signature 'DenseNeuroVol'
Summary(x, ..., na.rm = FALSE)

## S4 method for signature 'DenseNeuroVol'
Summary(x, ..., na.rm = FALSE)
```

Arguments

`x`	A neuroimaging object (SparseNeuroVec, SparseNeuroVol, or DenseNeuroVol)
`...`	Additional arguments passed to methods
`na.rm`	Logical indicating whether to remove NA values before computation

Value

The result of the summary operation

Examples

```
# Create a simple volume
vol <- DenseNeuroVol(array(1:27, c(3,3,3)),
                     NeuroSpace(c(3L,3L,3L), c(1,1,1)))
sum(vol)
range(vol)
```

`theme_neuro` *A minimal, publication-friendly theme for image slices*

Description

Quiet axes, thin panel border, no grid, generous margins, slim legend.

Usage

```
theme_neuro(base_size = 10, base_family = "")
```

Arguments

`base_size`	Base font size.
`base_family`	Base font family.

`TIME` *Time axis*

Description

Represents the temporal dimension in neuroimaging data

Usage

```
TIME
```

Format

An object of class `NamedAxis` of length 1.

`TimeAxis` *Time axis set*

Description

A one-dimensional axis set representing time

Usage

```
TimeAxis
```

Format

An object of class `AxisSet1D` of length 1.

`trans` *Extract image coordinate transformation*

Description

Extract image coordinate transformation

Get transformation matrix

Usage

```
trans(x)

## S4 method for signature 'MetaInfo'
trans(x)

## S4 method for signature 'NeuroObj'
trans(x)

## S4 method for signature 'NeuroSpace'
trans(x)
```

Arguments

`x`	an object with a transformation

Details

This function returns a transformation that can be used to go from "grid coordinates" to "real world coordinates" in millimeters. see `NeuroSpace`

Value

A numeric 4x4 matrix that maps from grid coordinates to real-world coordinates.

Examples

```
bspace <- NeuroSpace(c(10,10,10), c(2,2,2))
trans(bspace)
all.equal(dim(trans(bspace)), c(4,4))
```

`values` *Extract Data Values of an Object*

Description

Extract Data Values of an Object

Usage

```
values(x, ...)

## S4 method for signature 'ClusteredNeuroVec'
values(x)

## S4 method for signature 'DenseNeuroVol'
values(x)
```

```
## S4 method for signature 'SparseNeuroVol'
values(x)

## S4 method for signature 'ROIVol'
values(x, ...)

## S4 method for signature 'ROIVec'
values(x, ...)
```

Arguments

`x`	the object to get values from
`...`	additional arguments

Value

A vector or array containing the values extracted from `x`.

Examples

```
x <- NeuroSpace(c(10,10,10), c(1,1,1))
vol <- NeuroVol(rnorm(10 * 10 * 10), x)
values(vol)
```

`vectors` *Extract an ordered list of 1D vectors.*

Description

This function extracts an ordered list of 1D vectors from an object that supplies vector data. The `subset` argument specifies the subset of vectors to extract, and can be a vector of indices or a logical vector. The return value is a list containing the extracted vectors in the same order as the specified indices.

Usage

```
vectors(x, subset, ...)

## S4 method for signature 'NeuroVec,missing'
vectors(x)

## S4 method for signature 'DenseNeuroVec,missing'
vectors(x)

## S4 method for signature 'NeuroVec,numeric'
vectors(x, subset)
```

```
## S4 method for signature 'NeuroVec,logical'
vectors(x, subset)

## S4 method for signature 'NeuroVecSeq,missing'
vectors(x)

## S4 method for signature 'NeuroVecSeq,numeric'
vectors(x, subset)

## S4 method for signature 'NeuroVecSeq,logical'
vectors(x, subset)

## S4 method for signature 'ROIVec,missing'
vectors(x)

## S4 method for signature 'matrix,missing'
vectors(x)

## S4 method for signature 'ROIVec,integer'
vectors(x, subset)

## S4 method for signature 'matrix,integer'
vectors(x, subset)

## S4 method for signature 'matrix,numeric'
vectors(x, subset)

## S4 method for signature 'ROIVec,numeric'
vectors(x, subset)

## S4 method for signature 'ROIVec,logical'
vectors(x, subset)

## S4 method for signature 'SparseNeuroVec,missing'
vectors(x, nonzero = FALSE)
```

Arguments

`x`	the object that supplies the vector data.
`subset`	the subset of vectors to extract.
`...`	additional arguments to be passed to methods.
`nonzero`	only include nonzero vectors in output list

Value

A `list` containing the extracted vectors from `x` in the same order as `subset`.

A deflist object where each element is a function that returns the time series for a voxel. The length of the deflist equals the total number of voxels.

Examples

```
file_name <- system.file("extdata", "global_mask_v4.nii", package="neuroim2")
vec <- read_vec(file_name)
v <- vectors(vec)
mean(v[[1]])
```

`vec_from_vols` *Create NeuroVec from list of NeuroVol objects*

Description

Factory function to create a NeuroVec object from a list of NeuroVol objects. This is a convenience wrapper around the NeuroVec constructor that combines multiple 3D volumes into a single 4D NeuroVec.

Usage

```
vec_from_vols(vols, mask = NULL)
```

Arguments

`vols`	A list of `NeuroVol` objects. All volumes must have identical spatial dimensions.
`mask`	An optional logical array or `LogicalNeuroVol` object defining the subset of voxels to include. If provided, a SparseNeuroVec will be created.

Value

A `NeuroVec` object (either DenseNeuroVec or SparseNeuroVec depending on whether a mask is provided).

See Also

`NeuroVec`, `NeuroVol`

Examples

```
# Create a simple NeuroVec from list of volumes
spc <- NeuroSpace(c(10, 10, 10))
vol1 <- NeuroVol(rnorm(10*10*10), spc)
vol2 <- NeuroVol(rnorm(10*10*10), spc)
vec <- vec_from_vols(list(vol1, vol2))
print(dim(vec))  # Should be c(10, 10, 10, 2)
```

`vols` *Extract an ordered series of 3D volumes.*

Description

This function extracts an ordered series of 3D volumes from an object that supplies volume data. The `indices` argument specifies the subset of volumes to extract, and can be a vector of indices or a logical vector. The return value is a list containing the extracted volumes in the same order as the specified indices.

Usage

```
vols(x, indices, ...)

## S4 method for signature 'NeuroVec,numeric'
vols(x, indices)

## S4 method for signature 'NeuroVec,missing'
vols(x)
```

Arguments

`x`	the object that supplies the volume data.
`indices`	the subset of volumes to extract.
`...`	additional arguments to be passed to methods.

Value

A `list` containing the extracted 3D volumes from `x` in the same order as `indices`.

Examples

```
vec <- read_vec(system.file("extdata", "global_mask_v4.nii", package="neuroim2"))
vs <- vols(vec)
length(vs) == dim(vec)[4]

vs <- vols(vec, indices=1:3)
length(vs) == 3
```

`voxels` *extract voxel coordinates*

Description

extract voxel coordinates

Usage

```
voxels(x, ...)

## S4 method for signature 'Kernel'
voxels(x, center_voxel = NULL)
```

Arguments

`x`	the object to extract voxels from
`...`	additional arguments to function
`center_voxel`	the absolute location of the center of the voxel, default is (0,0,0)

Value

A `matrix` or `vector` representing voxel coordinates from `x`.

Examples

```
# Create a 3D kernel with dimensions 3x3x3 and voxel size 1x1x1
kern <- Kernel(kerndim = c(3,3,3), vdim = c(1,1,1))

# Get voxel coordinates centered at origin (0,0,0)
vox <- voxels(kern)
# Returns a matrix where each row is a voxel coordinate
# relative to the kernel center

# Get voxel coordinates centered at specific point (5,5,5)
vox_centered <- voxels(kern, center_voxel = c(5,5,5))
# Returns coordinates shifted to be centered at (5,5,5)
```

`which_dim` *Find Dimensions of a Given Axis*

Description

This function returns the dimension of the specified axis for a given object, such as a matrix or an array.

Usage

```
which_dim(x, axis)

## S4 method for signature 'NeuroSpace,NamedAxis'
which_dim(x, axis)
```

Arguments

`x`	The NeuroSpace object
`axis`	The NamedAxis to find

Value

An integer representing the dimension index of the specified `axis` for the object `x`.

Examples

```
x <- NeuroSpace(c(10,10,10), spacing=c(1,1,1))
which_dim(x, x@axes@j) == 2
```

`write_elements` *Write a sequence of elements from an input source*

Description

Write a sequence of elements from an input source

Usage

```
write_elements(x, els)

## S4 method for signature 'BinaryWriter,numeric'
write_elements(x, els)
```

Arguments

`x`	the output channel
`els`	the elements to write

Value

Invisibly returns `NULL` after writing the elements.

Examples

```
# Create a temporary binary file for writing
tmp <- tempfile()
writer <- BinaryWriter(tmp, byte_offset = 0L,
                      data_type = "DOUBLE", bytes_per_element = 8L)

# Write some random data
data <- rnorm(100)
write_elements(writer, data)
close(writer)

# Read back the data to verify
reader <- BinaryReader(tmp, byte_offset = 0L,
                      data_type = "double", bytes_per_element = 8L)
read_data <- read_elements(reader, 100)
close(reader)

# Verify data was written correctly
all.equal(data, read_data)

# Clean up
unlink(tmp)

# Create a temporary binary file for writing
tmp <- tempfile()
writer <- BinaryWriter(tmp, byte_offset = 0L,
                      data_type = "DOUBLE", bytes_per_element = 8L)
# Write some data
write_elements(writer, rnorm(100))
close(writer)

# Clean up
unlink(tmp)
```

`write_vec` *Write a 4d image vector to disk*

Description

Write a 4d image vector to disk

Usage

```
write_vec(x, file_name, format, data_type, ...)

## S4 method for signature 'ROIVec,character,missing,missing'
write_vec(x, file_name)

## S4 method for signature 'NeuroVec,character,missing,missing'
write_vec(x, file_name)

## S4 method for signature 'NeuroVec,character,character,missing'
write_vec(
  x,
  file_name,
  format,
  nbit = FALSE,
  compression = 5,
  chunk_dim = c(10, 10, 10, dim(x)[4])
)

## S4 method for signature 'NeuroVec,character,missing,character'
write_vec(x, file_name, data_type)

## S4 method for signature 'ROIVec,character,missing,missing'
write_vec(x, file_name)

## S4 method for signature 'NeuroVec,character,missing,missing'
write_vec(x, file_name)

## S4 method for signature 'NeuroVec,character,character,missing'
write_vec(
  x,
  file_name,
  format,
  nbit = FALSE,
  compression = 5,
  chunk_dim = c(10, 10, 10, dim(x)[4])
)

## S4 method for signature 'NeuroVec,character,missing,character'
write_vec(x, file_name, data_type)
```

Arguments

`x`	an image object, typically a `NeuroVec` instance.
`file_name`	output file name.
`format`	file format string. Since "NIFTI" is the only currently supported format, this parameter can be safely ignored and omitted.

`data_type`	the numeric data type. If specified should be a `character` vector of: "BINARY", "UBYTE", "SHORT", "INT", "FLOAT", "DOUBLE". Otherwise output format will be inferred from R the datatype of the image.
`...`	extra args
`nbit`	set nbit compression
`compression`	compression level 1 to 9
`chunk_dim`	the dimensions of each chunk

Value

Invisibly returns `NULL` after writing the vector to disk.

Examples

```
bvec <- NeuroVec(array(0, c(10,10,10,10)), NeuroSpace(c(10,10,10,10), c(1,1,1)))

# Create temporary files
tmp1 <- tempfile(fileext = ".nii")

# Write vectors to temporary files
write_vec(bvec, tmp1)

# Clean up
unlink(tmp1)
```

`write_vol` *Write a 3d image volume to disk*

Description

Write a 3d image volume to disk

Usage

```
write_vol(x, file_name, format, data_type)

## S4 method for signature 'NeuroVol,character,missing,missing'
write_vol(x, file_name)

## S4 method for signature 'ClusteredNeuroVol,character,missing,missing'
write_vol(x, file_name)

## S4 method for signature 'NeuroVol,character,character,missing'
write_vol(x, file_name, format)

## S4 method for signature 'ROIVol,character,character,missing'
```

```
write_vol(x, file_name, format)

## S4 method for signature 'NeuroVol,character,missing,character'
write_vol(x, file_name, data_type)
```

Arguments

`x`	an image object, typically a `NeuroVol` instance.
`file_name`	output file name
`format`	file format string. Since "NIFTI" is the only currently supported format, this parameter can be safely ignored and omitted.
`data_type`	output data type, If specified should be a `character` vector of: "BINARY", "UBYTE", "SHORT", "INT", "FLOAT", "DOUBLE". Otherwise output format will be inferred from R the datatype of the image.

Details

The output format will be inferred from file extension.

The output format will be inferred from file extension. `write_vol(x, "out.nii")` outputs a NIFTI file. `write_vol(x, "out.nii.gz")` outputs a gzipped NIFTI file.

No other file output formats are currently supported.

Value

Invisibly returns `NULL` after writing the volume to disk.

Examples

```
bvol <- NeuroVol(array(0, c(10,10,10)), NeuroSpace(c(10,10,10), c(1,1,1)))

tmp1 <- tempfile(fileext = ".nii")
write_vol(bvol, tmp1)
unlink(tmp1)
```

`[,AbstractSparseNeuroVec,numeric,numeric,ANY-method`
Extractor Method for AbstractSparseNeuroVec

Description

Extracts a subset of data from a sparse four-dimensional brain image based on provided indices.

Usage

```
## S4 method for signature 'AbstractSparseNeuroVec,numeric,numeric,ANY'
x[i, j, k, m, ..., drop = TRUE]
```

Arguments

`x`	An object of class `AbstractSparseNeuroVec`
`i`	Numeric vector specifying the indices for the first dimension
`j`	Numeric vector specifying the indices for the second dimension
`k`	Numeric vector specifying the indices for the third dimension (optional)
`m`	Numeric vector specifying the indices for the fourth dimension (optional)
`...`	Additional arguments passed to methods
`drop`	Logical indicating whether to drop dimensions of length one (default: TRUE)

Value

An array containing the extracted subset

`[,DenseNeuroVol,numeric,missing,ANY-method`

Extract or replace parts of an object

Description

Extract or replace parts of an object

Usage

```
## S4 method for signature 'DenseNeuroVol,numeric,missing,ANY'
x[i, j, k, ..., drop = TRUE]

## S4 method for signature 'DenseNeuroVol,integer,missing,ANY'
x[i, j, k, ..., drop = TRUE]

## S4 method for signature 'NeuroVol,ROIVol,missing,ANY'
x[i, j, k, ..., drop = TRUE]

## S4 method for signature 'DenseNeuroVol,ROIVol,missing,ANY'
x[i, j, k, ..., drop = TRUE]

## S4 method for signature 'ROIVol,numeric,missing,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'ROIVol,logical,missing,ANY'
x[i, j, k, ..., drop = TRUE]
```

Arguments

`x`	The object to extract from
`i`	Index specifying elements to extract
`j`	Second index (if applicable)
`k`	Third index for 3D objects (if applicable)
`...`	Additional arguments passed to methods
`drop`	Whether to drop dimensions of length 1

Value

A subset of the input object, with dimensions depending on the indexing and the ‘drop‘ parameter.

`[[,NeuroVec,numeric-method`

[[

Description

This function extracts a single volume from a NeuroVec object.

Usage

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

Arguments

`x`	The NeuroVec object.
`i`	The volume index to extract.

Value

a DenseNeuroVol object

`[[,NeuroVecSeq,numeric-method` *Extract Element from NeuroVecSeq*

Description

Extracts a single volume from a NeuroVecSeq object at the specified time point.

Usage

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

Arguments

`x`	A NeuroVecSeq object
`i`	Numeric index specifying the time point to extract

Value

A NeuroVol object representing the extracted volume

`[[,SparseNeuroVec,numeric-method` *[[*

Description

[[

Usage

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

Arguments

`x`	the object
`i`	the volume index

Value

a SparseNeuroVol object

Index