Package: fmrireg (via r-universe)

June 7, 2026

Type Package

Title Regression Analysis of Functional Magnetic Resonance Imaging Data

Version 0.2.0

LinkingTo Rcpp,RcppArmadillo, roptim, RcppParallel

License MIT + file LICENSE

Description Provides tools for the analysis of functional magnetic resonance imaging (fMRI) data. Facilities are included to construct flexible hemodynamic response functions, experimental regressors, and conduct univariate fMRI regression analyses.

Encoding UTF-8

LazyData false

Depends R (>= 4.0.0)

Imports fmriAR, fmrigds, fmridesign (>= 0.1.0), fmrihrf (>= 0.1.0), fmrilss (>= 0.1.0), stats, Matrix, MASS, RANN, assertthat, tibble, purrr, foreach, dplyr, ggplot2, lazyeval, tidyr, robustbase, matrixStats, furrr, rlang, neuroim2, Rcpp, RcppParallel, methods, fmridataset (>= 0.1.0), crayon, plotly, cli, bslib, future.apply, jsonlite, progress, magrittr

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE, roclets = c(` `namespace", ` `rd", ` `roxyglobals::global_roclet"))

Suggests testthat, RNifti, ggrepel, pls, doParallel, fmristore, knitr, rmarkdown, roxygen2, covr, gridExtra, multivarious, glmnet, proxy, cowplot, forecast, future, shiny, mgcv, progressr, pkgload, roxyglobals (>= 0.2.1), withr

Remotes bbuchsbaum/fmristore, bbuchsbaum/fmrigds@b131500b8e91e116c76f5d0df2a91cb7ecb4de14, anthonynorth/roxyglobals, bbuchsbaum/fmridataset, bbuchsbaum/fmrilss, bbuchsbaum/fmridesign, bbuchsbaum/neuroatlas, bbuchsbaum/delarr, bbuchsbaum/fmrilatent

VignetteBuilder knitr

URL https://bbuchsbaum.github.io/fmrireg/,
https://github.com/bbuchsbaum/fmrireg

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

Config/Needs/website albersdown

Config/pak/sysreqs cmake make libhdf5-dev libicu-dev libuv1-dev libssl-dev

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

Date/Publication 2026-06-07 00:47:33 UTC

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

RemoteRef HEAD

RemoteSha 64e6214da8082e437763d76267f6fd3d5dece45c

Contents

`.fmrireg_engine_registry` *Plugin registration and helpers for fmrireg*

Description

Plugin registration and helpers for fmrireg

Usage

```
.fmrireg_engine_registry
```

Format

An object of class `environment` of length 2.

`.resample_param` *Resample parameter vector with specified distribution*

Description

Helper for drawing per-event amplitudes/durations around base values.

Usage

```
.resample_param(
  base,
  sd,
  dist = c("lognormal", "gamma", "gaussian"),
  allow_negative = FALSE
)
```

Arguments

`base`	Numeric vector of base values to jitter
`sd`	Numeric standard deviation for the sampling distribution
`dist`	Distribution name: "lognormal", "gamma", or "gaussian"
`allow_negative`	Logical; if TRUE, allow negative draws (only used for gaussian)

Value

A numeric vector of resampled values with the same length as base

---

`apply_soft_projection` *Apply Soft Projection to Data and Design*

Description

Applies the soft projection to both the data matrix Y and design matrix X. Both must be projected to avoid bias in coefficient estimates.

Usage

```
apply_soft_projection(proj, Y, X)
```

Arguments

`proj`	A soft_projection object from `soft_projection()`.
`Y`	Data matrix (time x voxels).
`X`	Design matrix (time x predictors).

Value

List with projected Y and X.

Examples

```
set.seed(123)
N <- matrix(rnorm(100 * 20), nrow = 100, ncol = 20)
Y <- matrix(rnorm(100 * 50), nrow = 100, ncol = 50)
X <- cbind(1, rnorm(100))

proj <- soft_projection(N, lambda = "auto")
cleaned <- apply_soft_projection(proj, Y, X)
# Use cleaned$Y and cleaned$X for GLM fitting
```

ar_parameters *Extract Estimated AR Parameters from fmri_lm Fit*

Description

Retrieves the estimated autoregressive parameters from a fitted fMRI linear model that used AR error modeling.

Usage

```
ar_parameters(object, ...)

## S3 method for class 'fmri_lm'
ar_parameters(object, scope = c("average", "per_run", "raw"), ...)
```

Arguments

object	An object of class fmri_lm
...	Additional arguments (currently unused)
scope	Character; "average" (default) returns the pooled average AR coefficients, "per_run" returns a list of the run-level estimates, and "raw" returns the stored structure without post-processing.

Value

Depending on scope, either a numeric vector of averaged AR coefficients, a list of per-run coefficient vectors, or the raw stored structure. Returns NULL when no AR modeling was performed.

Examples

```
## Not run:
# Fit model with AR(1) errors
fit <- fmri_lm(onset ~ hrf(cond), dataset = dset, cor_struct = "ar1")
ar_parameters(fit)  # Extract estimated AR(1) coefficient

## End(Not run)
```

as.array.NeuroVec *Coerce NeuroVec to base array*

Description

This method allows NeuroVec objects from neuroim2 to be converted to base R arrays using the standard as.array() function. This is particularly useful in testing and data manipulation contexts.

Usage

```
## S3 method for class 'NeuroVec'
as.array(x, ...)
```

Arguments

x	A NeuroVec object from neuroim2
...	Additional arguments (currently unused)

Value

A base R array containing the data from the NeuroVec object

---

autoplot.Reg *Autoplot method for Reg objects*

Description

Creates a ggplot visualization of an fMRI regressor object.

Usage

```
## S3 method for class 'Reg'
autoplot(object, grid = NULL, precision = 0.1, method = "conv", ...)
```

Arguments

object	A Reg object (or one inheriting from it, like regressor).
grid	Optional numeric vector specifying time points (seconds) for evaluation. If NULL, a default grid is generated based on the object's onsets and span.
precision	Numeric precision for HRF evaluation if grid needs generation or if internal evaluation requires it (passed to evaluate).
method	Evaluation method passed to evaluate.
...	Additional arguments (currently unused).

Value

A ggplot object.

blockids.event_model *Block IDs for event_model*

Description

Return the run/block IDs associated with an event_model's sampling frame.

Usage

```
## S3 method for class 'event_model'
blockids(x, ...)
```

Arguments

x	An event_model object
...	Additional arguments passed through

Value

Integer vector of block IDs

Examples

```
ev <- fmrireg:::.demo_event_model()
blockids(ev)
```

coef.fmri_meta *Extract Coefficients from Meta-Analysis*

Description

Extract Coefficients from Meta-Analysis

Usage

```
## S3 method for class 'fmri_meta'
coef(object, ...)
```

Arguments

object	An fmri_meta object
...	Additional arguments

Value

Matrix of coefficients

Examples

```
toy_meta <- structure(
  list(coefficients = matrix(c(0.2, -0.1), nrow = 1)),
  class = "fmri_meta"
)
coef(toy_meta)
```

`coef_image.fmri_lm` *Extract Image/Volume for Coefficient*

Description

Extract Image/Volume for Coefficient

Usage

```
## S3 method for class 'fmri_lm'
coef_image(
  object,
  coef = 1,
  statistic = c("estimate", "se", "tstat", "prob"),
  type = c("estimates", "contrasts", "F"),
  ...
)

coef_image(object, coef = 1, statistic = c("estimate", "se", "z", "p"), ...)
```

Arguments

`object`	An fmri_meta object
`coef`	Coefficient name or index
`statistic`	Type of statistic to extract ("estimate", "se", "z", "p")
`type`	For `fmri_lm` objects: which coefficient set to index into: `"estimates"` (default), `"contrasts"`, or `"F"`.
`...`	Additional arguments (currently unused).

Value

NeuroVol object or matrix

Examples

```
# Create a small example
X <- matrix(rnorm(50 * 4), 50, 4)
edata <- data.frame(
  condition = factor(c("A", "B", "A", "B")),
  onsets = c(1, 12, 25, 38),
  run = c(1, 1, 1, 1)
)
dset <- fmridataset::matrix_dataset(X, TR = 2, run_length = 50,
                                    event_table = edata)
fit <- fmri_lm(onsets ~ hrf(condition), block = ~run, dataset = dset)
# Get coefficient estimates as a numeric vector
coef_image(fit, coef = 1)
toy_meta <- structure(
  list(
    coefficients = matrix(c(0.3, 0.1), nrow = 1,
      dimnames = list(NULL, c("A", "B"))),
    se = matrix(c(0.05, 0.06), nrow = 1)
  ),
  class = "fmri_meta"
)
coef_image(toy_meta, coef = "A")
```

`coef_image.fmri_ttest_fit` *Extract Coefficient Image from fmri_ttest_fit*

Description

Creates a NeuroVol image from coefficients in an fmri_ttest_fit object.

Usage

```
## S3 method for class 'fmri_ttest_fit'
coef_image(object, coef = 1, statistic = c("estimate", "se", "z", "p"), ...)
```

Arguments

`object`	An fmri_ttest_fit object
`coef`	Character or integer; coefficient to extract
`statistic`	Character string; type of statistic to extract: • "estimate": Coefficient estimates (beta values) • "se": Standard errors (if available) • "z": Z-scores • "p": P-values
`...`	Additional arguments (e.g., mask to apply)

Value

NeuroVol object or numeric vector

`coef_images.fmri_lm` *Extract Images/Volumes for Multiple Coefficients*

Description

A plural companion to `coef_image`. Returns a named list of volumes (or numeric vectors for non-spatial datasets), one per coefficient, so callers can iterate over or write out every coefficient without reimplementing the loop over `coef_names`.

Usage

```
## S3 method for class 'fmri_lm'
coef_images(
  object,
  statistic = c("estimate", "se", "tstat", "prob"),
  type = c("estimates", "contrasts", "F"),
  coefs = NULL,
  ...
)

coef_images(object, ...)
```

Arguments

`object`	A fitted model object.
`statistic`	For `fmri_lm` objects: one of `"estimate"`, `"se"`, `"tstat"`, or `"prob"`.
`type`	For `fmri_lm` objects: which coefficient set to extract: `"estimates"` (default), `"contrasts"`, or `"F"`.
`coefs`	Optional character vector of coefficient names (a subset of `coef_names(object, type = type)`) to extract. Defaults to all coefficients of the requested `type`.
`...`	Additional arguments passed to methods (and on to `coef_image`).

Value

A named list of `NeuroVol` objects (or numeric vectors when the dataset is non-spatial), named by coefficient.

See Also

`coef_image`, `coef_names`

Examples

```
# Create a small example
X <- matrix(rnorm(50 * 4), 50, 4)
edata <- data.frame(
  condition = factor(c("A", "B", "A", "B")),
  onsets = c(1, 12, 25, 38),
  run = c(1, 1, 1, 1)
)
dset <- fmridataset::matrix_dataset(X, TR = 2, run_length = 50,
                                    event_table = edata)
fit <- fmri_lm(onsets ~ hrf(condition), block = ~run, dataset = dset)
# Named list of one volume per event regressor
imgs <- coef_images(fit, statistic = "estimate", type = "estimates")
names(imgs)
```

---

coef_names *Get Available Coefficient Names*

Description

Return the names of available coefficients from a fitted model object. This helps users discover which coefficient names can be passed to coef_image or other extraction functions.

Usage

```
coef_names(x, ...)

## S3 method for class 'fmri_lm'
coef_names(x, type = c("estimates", "contrasts", "F", "all"), ...)
```

Arguments

x	A fitted model object
...	Additional arguments passed to methods
type	Which set of names to return: "estimates" (default) for event regressor names, "contrasts" for simple contrast names, "F" for F-contrast names, or "all" for a named list of all three.

Value

A character vector of coefficient names

See Also

coef_image, coef

Other statistical_measures: p_values(), standard_error(), stats()

Examples

```
# Create a small example
X <- matrix(rnorm(50 * 4), 50, 4)
edata <- data.frame(
  condition = factor(c("A", "B", "A", "B")),
  onsets = c(1, 12, 25, 38),
  run = c(1, 1, 1, 1)
)
dset <- fmridataset::matrix_dataset(X, TR = 2, run_length = 50,
                                    event_table = edata)
fit <- fmri_lm(onsets ~ hrf(condition), block = ~run, dataset = dset)
coef_names(fit)
```

`columns` *Extract Column Names or Identifiers*

Description

Extract column names or identifiers from an object. For parametric basis objects, this returns tokens representing the type of variables (categorical or continuous).

Usage

```
columns(x, ...)

## S3 method for class 'event_model'
columns(x, ...)
```

Arguments

`x`	An object (typically a ParametricBasis)
`...`	Additional arguments passed to method-specific implementations.

Value

A character vector of column identifiers

Examples

```
dat <- data.frame(
  onsets = c(0, 4, 8),
  condition = factor(c("A", "B", "A")),
  run = 1
)
ev <- event_model(
  onsets ~ hrf(condition),
  data = dat,
  block = ~ run,
```

```
  sampling_frame = fmrihrf::sampling_frame(blocklens = 12, TR = 2)
)
columns(ev)
```

`compute_dvars` *Compute DVARS (Temporal Derivative of Timecourses)*

Description

DVARS measures the root mean square of the temporal derivative across all voxels, providing a single quality metric per volume. High DVARS indicates rapid signal changes, often associated with motion artifacts.

Usage

```
compute_dvars(Y, normalize = TRUE)
```

Arguments

`Y`	Numeric matrix of fMRI data (time x voxels).
`normalize`	Logical. If TRUE, normalize by median DVARS. Default TRUE.

Value

Numeric vector of length nrow(Y) with DVARS values. The first timepoint is set to the median of subsequent values.

Examples

```
# Simulate fMRI data with a motion spike
set.seed(123)
Y <- matrix(rnorm(100 * 50), nrow = 100, ncol = 50)
Y[50, ] <- Y[50, ] + 5  # Add spike at volume 50
dvars <- compute_dvars(Y)
# dvars[50] will be elevated compared to other volumes

# Identify suspect volumes (normalized DVARS > 1.5)
suspect <- which(dvars > 1.5)
cat("Suspect volumes:", suspect, "\n")
```

`compute_lm_contrasts` *Compute linear model contrast statistics (public API)*

Description

A polished wrapper around the internal fast contrast engine. Accepts name-based contrast specifications and returns a consistent tibble by default.

Usage

```
compute_lm_contrasts(
  B,
  XtXinv,
  df,
  sigma = NULL,
  sigma2 = NULL,
  contrasts = NULL,
  t_contrasts = NULL,
  f_contrasts = NULL,
  columns = NULL,
  output = c("stacked", "list"),
  robust_weights = NULL,
  ar_order = 0,
  drop_failed = TRUE
)
```

Arguments

`B`	Numeric matrix (p x V) of coefficients.
`XtXinv`	Numeric matrix (p x p), inverse cross-product of the design.
`df`	Residual degrees of freedom.
`sigma`	Optional numeric vector (length V) of residual std. dev.; ignored if `sigma2` provided.
`sigma2`	Optional numeric vector (length V) of residual variances.
`contrasts`	Optional named list mixing t- and F-contrasts; vectors are t, matrices are F.
`t_contrasts`	Optional named list of numeric vectors (t-contrasts).
`f_contrasts`	Optional named list of numeric matrices (F-contrasts).
`columns`	Optional character vector (length p) naming coefficients; used for name matching.
`output`	Either "stacked" (default; tibble) or "list" (raw list of tibbles).
`robust_weights`	Optional numeric vector of robust weights or NULL. Used only for df adjustments.
`ar_order`	Integer AR order; used only for effective df adjustments.
`drop_failed`	Logical; drop contrasts that fail validation (default TRUE).

Details

Inputs mirror a standard GLM contrast computation: provide `B` (p x V), `XtXinv` (p x p), residual degrees of freedom `df`, and either `sigma` or `sigma2` (per-voxel noise). Contrasts can be specified as a single mixed `contrasts` list or separately as `t_contrasts` (vectors) and `f_contrasts` (matrices). When contrast vectors/matrices are named (for vectors) or have column names (for matrices), names are matched to `columns` to determine the appropriate design indices; otherwise lengths must match the number of parameters `p`.

Value

A tibble with rows per contrast (default) or a named list if `output = "list"`.

`compute_lm_contrasts_from_suffstats` *Compute contrast statistics from sufficient statistics (public API)*

Description

Convenience wrapper that accepts design/data sufficient statistics, computes betas and residual variance, and delegates to `compute_lm_contrasts()`.

Usage

```
compute_lm_contrasts_from_suffstats(
  XtX,
  XtS,
  StS,
  df,
  sigma = NULL,
  sigma2 = NULL,
  contrasts = NULL,
  t_contrasts = NULL,
  f_contrasts = NULL,
  columns = NULL,
  output = c("stacked", "list"),
  robust_weights = NULL,
  ar_order = 0,
  drop_failed = TRUE
)
```

Arguments

`XtX`	Numeric (p x p) cross-product of the design.
`XtS`	Numeric (p x V) cross-product of design with data.
`StS`	Numeric length-V vector of sum of squares per voxel.
`df`	Residual degrees of freedom.

sigma	Optional numeric vector (length V) of residual std. dev.; ignored if sigma2 provided.
sigma2	Optional numeric vector (length V) of residual variances.
contrasts	Optional named list mixing t- and F-contrasts; vectors are t, matrices are F.
t_contrasts	Optional named list of numeric vectors (t-contrasts).
f_contrasts	Optional named list of numeric matrices (F-contrasts).
columns	Optional character vector (length p) naming coefficients; used for name matching.
output	Either "stacked" (default; tibble) or "list" (raw list of tibbles).
robust_weights	Optional numeric vector of robust weights or NULL. Used only for df adjustments.
ar_order	Integer AR order; used only for effective df adjustments.
drop_failed	Logical; drop contrasts that fail validation (default TRUE).

Value

A tibble with rows per contrast (default) or a named list if `output = "list"`.

`conditions.convolved_term`

Visualize the entire design matrix as a heatmap

Description

Generate a heatmap visualization of a design matrix, showing regressor values over time. This is useful for inspecting the temporal structure of fMRI design matrices.

Usage

```
## S3 method for class 'convolved_term'
conditions(x, ...)
```

Arguments

x	The model object (event_model, baseline_model, or fmri_model)
...	Additional arguments passed to methods. Common arguments include: **rescale_cols** Logical; if TRUE, columns are rescaled to (-1,1) **block_separators** Logical; if TRUE, draw white lines between blocks **rotate_x_text** Logical; if TRUE, rotate x-axis labels by 45 degrees

Value

A ggplot2 object containing the design matrix heatmap

contrast *Contrast generic*

Description

Provide a contrast generic that dispatches on the first argument. Falls back to fmridesign::contrast for non-fmri_meta classes.

Usage

```
contrast(x, ...)
```

Arguments

x	object
...	passed to methods

Value

A contrast object with computed contrast weights and statistics

Examples

```
meta <- fmrireg:::.demo_fmri_meta()
contrast(meta, c("(Intercept)" = 1))
```

contrast.fmri_meta *Apply Contrast to Meta-Analysis Results*

Description

Note: Uses exact standard errors when covariance is available (return_cov="tri") or for ROI CSV fits with non-robust estimation; otherwise uses a diagonal variance approximation.

Usage

```
## S3 method for class 'fmri_meta'
contrast(x, contrast, ...)
```

Arguments

x	An fmri_meta object
contrast	Contrast specification. Can be: • A numeric vector of contrast weights • A formula (e.g., ~ groupold - groupyoung) • A named vector (e.g., c("groupold" = 1, "groupyoung" = -1))
...	Additional arguments

Value

An fmri_meta_contrast object with contrast results

Examples

```
toy_meta <- structure(
  list(
    coefficients = matrix(c(0.3, 0.1), nrow = 1,
      dimnames = list(NULL, c("A", "B"))),
    se = matrix(c(0.05, 0.06), nrow = 1),
    robust = "none"
  ),
  class = "fmri_meta"
)
contrast(toy_meta, c(1, -1))
```

---

contrast_set *Create a contrast set*

Description

Create a contrast set

Usage

```
contrast_set(...)
```

Arguments

...	contrast specifications

Value

A list of class "contrast_set" containing the specified contrasts

Examples

```
cs <- contrast_set(
  fmridesign::pair_contrast(~condition == "A", ~condition == "B", name = "A_vs_B")
)
```

`correlation_map` *correlation_map*

Description

Generate a correlation heatmap showing the relationships between columns in a design matrix. This visualization helps identify potential collinearity between regressors in the model. For event models, it shows correlations between different conditions. For baseline models, it shows correlations between drift and nuisance terms.

These methods provide correlation heatmap visualizations for various model objects. They are thin wrappers around methods from fmridesign when appropriate.

Usage

```
correlation_map(x, ...)

## S3 method for class 'baseline_model'
correlation_map(
  x,
  method = c("pearson", "spearman"),
  half_matrix = FALSE,
  absolute_limits = TRUE,
  ...
)
```

Arguments

`x`	The model object (event_model, baseline_model, or fmri_model)
`...`	Additional arguments passed to methods. Common arguments include: `method` Correlation method: "pearson" (default) or "spearman" `half_matrix` Logical; if TRUE, show only lower triangle (default: FALSE) `absolute_limits` Logical; if TRUE, set color limits to [-1,1] (default: TRUE)
`method`	Correlation method: "pearson" (default) or "spearman"
`half_matrix`	Logical; if TRUE, show only lower triangle (default: FALSE)
`absolute_limits`	Logical; if TRUE, set color limits to [-1,1] (default: TRUE)

Details

Create a correlation heatmap for an fMRI design matrix.

Value

A ggplot2 object containing the correlation heatmap, where:

- Rows and columns represent model terms
- Colors indicate correlation strength (-1 to 1)
- Darker colors indicate stronger correlations

A ggplot2 object containing the correlation heatmap visualization

See Also

event_model(), baseline_model()

Examples

```
# Create event data
event_data <- data.frame(
  condition = factor(c("face", "house", "face", "house")),
  rt = c(0.8, 1.2, 0.9, 1.1),
  onsets = c(1, 10, 20, 30),
  run = c(1, 1, 1, 1)
)

# Create sampling frame
sframe <- sampling_frame(blocklens = 50, TR = 2)

# Create event model
evmodel <- event_model(
  onsets ~ hrf(condition) + hrf(rt),
  data = event_data,
  block = ~run,
  sampling_frame = sframe
)

# Plot correlation map for event model
correlation_map(evmodel)

# Create baseline model
bmodel <- baseline_model(
  basis = "bs",
  degree = 3,
  sframe = sframe
)

# Plot correlation map for baseline model
correlation_map(bmodel)

# Note: To create a full fmri_model and plot combined correlations,
# you would need an fmri_dataset object:
# fmodel <- fmri_model(evmodel, bmodel, dataset)
# correlation_map(fmodel, method = "pearson", half_matrix = TRUE)
```

correlation_map.fmri_model *correlation_map.fmri_model*

Description

Generates a correlation heatmap of the columns in an `fmri_model`'s combined event+baseline design matrix.

Usage

```
## S3 method for class 'fmri_model'
correlation_map(
  x,
  method = c("pearson", "spearman"),
  half_matrix = FALSE,
  absolute_limits = TRUE,
  ...
)
```

Arguments

`x`	An `fmri_model`.
`method`	Correlation method (e.g., "pearson", "spearman").
`half_matrix`	Logical; if TRUE, display only the lower triangle of the matrix.
`absolute_limits`	Logical; if TRUE, set color scale limits from -1 to 1.
`...`	Additional arguments passed to internal plotting functions.

create_3d_blocks *Create 3D Blocks for Voxelwise Analysis*

Description

Creates spatial blocks from a 3D mask for use with spatial_fdr. This is useful for voxelwise analyses where you want to group nearby voxels together for more powerful multiple comparisons correction.

Usage

```
create_3d_blocks(mask, block_size = c(10, 10, 10), connectivity = 26)
```

Arguments

mask	Numeric 3D array or NeuroVol object defining the brain mask. Non-zero values indicate voxels to include.
block_size	Integer vector of length 3 specifying block dimensions in voxels (default: c(10, 10, 10))
connectivity	Integer scalar; type of connectivity for neighbors: 6 (face connectivity) or 26 (face, edge, and corner connectivity). Default: 26

Value

List with components:

group_id	Integer vector of group IDs for each voxel in the mask
neighbors	List of length n_groups where element i contains integer vector of 1-based neighbor IDs for group i
n_groups	Integer scalar; total number of groups created
block_size	Block size used

Examples

```
mask <- array(c(1L, 1L, 0L,
                1L, 1L, 0L,
                0L, 0L, 0L), dim = c(3, 3, 1))
blocks <- create_3d_blocks(mask, block_size = c(2, 2, 1))
blocks$n_groups
```

---

create_design_matrix_from_benchmark

Create Design Matrix from Benchmark Dataset

Description

Helper function to create a design matrix from a benchmark dataset using a specified HRF. This is useful for testing different HRF assumptions against the ground truth.

Usage

```
create_design_matrix_from_benchmark(
  dataset_name,
  hrf,
  include_intercept = TRUE
)
```

Arguments

dataset_name	Character string specifying which dataset to use
hrf	HRF object to use for convolution (e.g., fmrihrf::HRF_SPMG1)
include_intercept	Logical, whether to include an intercept column

Value

A matrix with the design matrix (time x conditions)

Examples

```
# Create design matrix using canonical HRF
X <- create_design_matrix_from_benchmark("BM_Canonical_HighSNR", fmrihrf::HRF_SPMG1)

# Test with a different HRF
X_wrong <- create_design_matrix_from_benchmark("BM_Canonical_HighSNR", fmrihrf::HRF_SPMG2)
```

---

design_map.fmri_model *Heatmap visualization of the combined fmri_model design matrix*

Description

Produces a single heatmap of *all* columns in the design matrix from an fmri_model object, which merges both the event_model and baseline_model regressors. Rows are scans; columns are regressors. Optionally draws horizontal lines between blocks (runs), and rotates x-axis labels diagonally for readability.

Usage

```
## S3 method for class 'fmri_model'
design_map(
  x,
  block_separators = TRUE,
  rotate_x_text = TRUE,
  fill_midpoint = NULL,
  fill_limits = NULL,
  ...
)
```

Arguments

x	An fmri_model object.
block_separators	Logical; if TRUE, draw white horizontal lines between blocks.

rotate_x_text	Logical; if TRUE, rotate x-axis labels by 45 degrees.
fill_midpoint	Numeric or NULL; if not NULL, passed to scale_fill_gradient2 to center the color scale (e.g. fill_midpoint=0).
fill_limits	Numeric vector of length 2 or NULL; passed to the fill scale limits= argument. This can clip or expand the color range.
...	Additional arguments passed to geom_tile.

Value

A ggplot2 plot object.

`design_matrix.convolved_term`

Design Matrix for Convolved Terms

Description

Extract the design matrix from a convolved term object, optionally filtered by block ID.

Usage

```
## S3 method for class 'convolved_term'
design_matrix(x, blockid = NULL, ...)
```

Arguments

x	A convolved_term object
blockid	Optional numeric vector specifying which blocks/runs to include
...	Additional arguments (not used)

Value

A matrix containing the convolved design matrix

design_matrix.fmri_lm *Design Matrix Method for fmri_lm Objects*

Description

Extract the design matrix from an fmri_lm object by delegating to its model component.

Usage

```
## S3 method for class 'fmri_lm'
design_matrix(x, ...)
```

Arguments

x	An fmri_lm object
...	Additional arguments passed to the design_matrix method for the model

Value

The design matrix from the fmri_lm object's model

design_matrix.fmri_model *Design Matrix for fMRI Models*

Description

Extract the combined design matrix from an fMRI model containing both event and baseline terms.

Usage

```
## S3 method for class 'fmri_model'
design_matrix(x, blockid = NULL, ...)
```

Arguments

x	An fmri_model object
blockid	Optional numeric vector specifying which blocks/runs to include
...	Additional arguments (not used)

Value

A tibble containing the combined design matrix with event and baseline terms

Examples

```
fm <- fmrireg:::.demo_fmri_model()
head(design_matrix(fm))
```

`design_plot` *Design Plot for fMRI Model*

Description

Generates an interactive Shiny app that plots the design matrix for a given fMRI model. The design matrix is first converted into a long-format tibble and then plotted over time, faceted by block. Several customization options allow you to adjust the title, axis labels, line thickness, color palette, and more.

Usage

```
design_plot(
  fmrimod,
  term_name = NULL,
  longnames = FALSE,
  plot_title = NULL,
  x_label = "Time (s)",
  y_label = "Amplitude",
  line_size = 1,
  color_palette = "viridis",
  facet_ncol = 2,
 theme_custom = ggplot2::theme_minimal(base_size = 15) + ggplot2::theme(panel.spacing =
    ggplot2::unit(1, "lines")),
  legend_threshold = 30,
  ...
)
```

Arguments

`fmrimod`	An `fmri_model` object.
`term_name`	Optional: Name of the term to plot. If `NULL` (the default), the first term is used.
`longnames`	Logical; if TRUE, use long condition names in the legend. Default is FALSE.
`plot_title`	Optional plot title. If `NULL`, a default title is generated.
`x_label`	Label for the x-axis. Default is "Time".
`y_label`	Label for the y-axis. Default is "Value".
`line_size`	Numeric; line thickness for the plot. Default is 1.
`color_palette`	Character; name of a ColorBrewer palette to use (e.g., "Set1"). Default is "Set1".
`facet_ncol`	Number of columns for facet_wrap. Default is 1.
`theme_custom`	A ggplot2 theme to apply. Default is `theme_bw(base_size = 14)`.
`legend_threshold`	Numeric; if the number of unique conditions exceeds this value, the legend is hidden. Default is 25.
`...`	Additional arguments passed to ggplot2::geom_line().

Value

A Shiny app that displays the design plot.

Examples

```
if (interactive()) {
  ## --- Construct a sampling frame ---
  sframe <- fmrihrf::sampling_frame(blocklens = c(100, 100), TR = 2, precision = 0.5)

  ## --- Create a dummy event table ---
  set.seed(123)
  event_table <- data.frame(
    onset = seq(10, 190, length.out = 20),
    x = rnorm(20),
    y = rnorm(20),
    run = rep(1:2, each = 10)
  )

  ## --- Construct a baseline model ---
 base_mod <- baseline_model(basis = "bs", degree = 3, sframe = sframe, intercept = "runwise")

  ## --- Construct an event model using a formula ---
  ev_mod <- event_model(x = onset ~ hrf(x) + hrf(y), data = event_table,
                        block = ~ run, sampling_frame = sframe,
                        drop_empty = TRUE, durations = rep(0, nrow(event_table)))

  ## --- Combine into an fMRI model ---
  fmri_mod <- fmri_model(ev_mod, base_mod)

  ## --- Launch the interactive design plot ---
  design_plot(fmrimod = fmri_mod,
              term_name = NULL,
              longnames = TRUE,
              plot_title = "fMRI Design Matrix",
              x_label = "Time (s)",
              y_label = "Signal",
              line_size = 1.5,
              color_palette = "Set2",
              facet_ncol = 1)
}
```

`dvars_to_weights` *Convert DVARS to Volume Weights*

Description

Transforms DVARS quality metrics into weights for weighted least squares fitting. Volumes with high DVARS receive lower weights, implementing "soft scrubbing" without hard censoring thresholds.

Usage

```
dvars_to_weights(
  dvars,
  method = c("inverse_squared", "soft_threshold", "tukey"),
  threshold = 1.5,
  steepness = 2
)
```

Arguments

dvars	Numeric vector of DVARS values (from compute_dvars).
method	Character. Weighting method: "inverse_squared" (default), "soft_threshold", or "tukey".
threshold	Numeric. For "soft_threshold", the DVARS value above which weights decay. Default 1.5 (1.5x median if normalized).
steepness	Numeric. For "soft_threshold", controls decay rate. Default 2.

Value

Numeric vector of weights in the interval from 0 to 1, same length as dvars.

Examples

```
set.seed(123)
Y <- matrix(rnorm(100 * 50), nrow = 100, ncol = 50)
Y[50, ] <- Y[50, ] + 5
dvars <- compute_dvars(Y)

# Compare different weighting methods
w_inv <- dvars_to_weights(dvars, method = "inverse_squared")
w_soft <- dvars_to_weights(dvars, method = "soft_threshold")
w_tukey <- dvars_to_weights(dvars, method = "tukey")

# Check weight at the artifact volume
cat("Weights at volume 50:\n")
cat("  inverse_squared:", round(w_inv[50], 3), "\n")
cat("  soft_threshold:", round(w_soft[50], 3), "\n")
cat("  tukey:", round(w_tukey[50], 3), "\n")
```

---

engine_spec *Inspect a Registered Engine Specification*

Description

Returns a read-only description of a registered engine, including its normalized capabilities, source ("builtin" vs "plugin"), aliases, and dispatch strategy. This is intended for extension authors and diagnostic tooling; it does not expose the underlying fit/preflight functions.

Usage

```
engine_spec(name)
```

Arguments

name	Engine name, such as "rrr_gls" or "latent_sketch".

Value

A list of class fmrireg_engine_spec, or NULL if the engine is not registered.

---

engine_specs *List Registered Engine Specifications*

Description

Returns the read-only specifications for all currently registered engines.

Usage

```
engine_specs()
```

Value

A named list of fmrireg_engine_spec objects.

---

estimate *Deprecated helper: estimate()*

Description

This function is deprecated. Use estimate_betas() instead.

Usage

```
estimate(...)
```

Arguments

...	Ignored.

Value

No return value; always errors with a deprecation message.

`estimate_betas` *Estimate Beta Coefficients for fMRI Data*

Description

Estimate beta coefficients (regression parameters) from fMRI data using various methods. This function supports different estimation approaches for:

single Single-trial beta estimation

effects Fixed and random effects

regularization Various regularization techniques

hrf Optional HRF estimation

This function estimates betas (regression coefficients) for fixed and random effects in a matrix dataset using various methods.

Usage

```
estimate_betas(x, ...)

## S3 method for class 'latent_dataset'
estimate_betas(
  x,
  fixed = NULL,
  ran,
  block,
  method = c("mixed", "lss", "ols"),
  basemod = NULL,
  prewhiten = FALSE,
  progress = TRUE,
  ...
)
```

Arguments

`x`	An object of class `matrix_dataset` representing the matrix dataset
`...`	Additional arguments passed to the estimation method
`fixed`	A formula specifying the fixed regressors that model constant effects (i.e., non-varying over trials)
`ran`	A formula specifying the random (trialwise) regressors that model single trial effects
`block`	A formula specifying the block factor
`method`	The regression method for estimating trialwise betas; one of "mixed", "lss", or "ols" (default: "mixed")

basemod	A baseline_model instance to regress out of data before beta estimation (default: NULL)
prewhiten	currently experimental, default to FALSE.
progress	Logical; show progress bar.

Details

This is a generic function with methods for different dataset types:

fmri_dataset For volumetric fMRI data

matrix_dataset For matrix-format data

latent_dataset For dimensionality-reduced data

Available estimation methods include:

mixed Mixed-effects model using ridge/BLUP estimation

r1 Rank-1 GLM with joint HRF estimation

lss Least-squares separate estimation

pls Partial least squares regression

ols Ordinary least squares

Value

A list of class "fmri_betas" containing:

betas_fixed Fixed effect coefficients

betas_ran Random (trial-wise) coefficients

design_ran Design matrix for random effects

design_fixed Design matrix for fixed effects

design_base Design matrix for baseline model

method_specific Additional components specific to the estimation method used

A list of class "fmri_betas" containing the following components:

- betas_fixed: Matrix representing the fixed effect betas
- betas_ran: Matrix representing the random effect betas
- design_ran: Design matrix for random effects
- design_fixed: Design matrix for fixed effects
- design_base: Design matrix for baseline model

References

Mumford, J. A., et al. (2012). Deconvolving BOLD activation in event-related designs for multivoxel pattern classification analyses. NeuroImage, 59(3), 2636-2643.

Pedregosa, F., et al. (2015). Data-driven HRF estimation for encoding and decoding models. NeuroImage, 104, 209-220.

See Also

fmri_dataset, matrix_dataset, latent_dataset

matrix_dataset, baseline_model

Other estimate_betas: estimate_betas.matrix_dataset()

Examples

```
# Create example data
event_data <- data.frame(
  condition = factor(c("A", "B", "A", "B")),
  onset = c(1, 10, 20, 30),
  run = c(1, 1, 1, 1)
)

# Create sampling frame and dataset
sframe <- sampling_frame(blocklens = 100, TR = 2)
dset <- fmridataset::matrix_dataset(
  matrix(rnorm(100 * 2), 100, 2),
  TR = 2,
  run_length = 100,
  event_table = event_data
)

# Estimate betas using mixed-effects model
betas <- estimate_betas(
  dset,
  fixed = onset ~ hrf(condition),
  ran = onset ~ trialwise(),
  block = ~run,
  method = "mixed"
)
```

---

estimate_betas.fmri_dataset

Estimate betas using various regression methods

Description

This function estimates betas (regression coefficients) for fixed and random effects using various regression methods including mixed models, least squares, and PLS.

Usage

```
## S3 method for class 'fmri_dataset'
estimate_betas(
  x,
  fixed = NULL,
```

```
  ran,
  block,
  method = c("mixed", "lss", "ols"),
  basemod = NULL,
  maxit = 1000,
  fracs = 0.5,
  progress = TRUE,
  ...
)
```

Arguments

x	An object of class fmri_dataset representing the fMRI dataset.
fixed	A formula specifying the fixed regressors that model constant effects (i.e., non-varying over trials).
ran	A formula specifying the random (trialwise) regressors that model single trial effects.
block	A formula specifying the block factor.
method	The regression method for estimating trialwise betas; one of "mixed", "lss", or "ols".
basemod	A baseline_model instance to regress out of data before beta estimation (default: NULL).
maxit	Maximum number of iterations for optimization methods (default: 1000).
fracs	Fraction of voxels used for prewhitening.
progress	Logical; show progress bar.
...	Additional arguments passed to the estimation method.

Value

A list of class "fmri_betas" containing the following components:

- betas_fixed: NeuroVec object representing the fixed effect betas.
- betas_ran: NeuroVec object representing the random effect betas.
- design_ran: Design matrix for random effects.
- design_fixed: Design matrix for fixed effects.
- design_base: Design matrix for baseline model.
- basemod: Baseline model object.
- fixed_model: Fixed effect model object.
- ran_model: Random effect model object.
- estimated_hrf: The estimated HRF vector (NULL for most methods).

See Also

fmri_dataset, baseline_model, event_model

Examples

```
## Not run:
facedes <- read.table(system.file("extdata", "face_design.txt", package = "fmrireg"), header=TRUE)
facedes$frun <- factor(facedes$run)
scans <- paste0("rscan0", 1:6, ".nii")

dset <- fmri_dataset(scans=scans, mask="mask.nii", TR=1.5,
        run_length=rep(436,6), event_table=facedes)
fixed = onset ~ hrf(run)
ran = onset ~ trialwise()
block = ~ run

betas <- estimate_betas(dset, fixed=fixed, ran=ran, block=block, method="mixed")

## End(Not run)
```

`estimate_betas.matrix_dataset` *Estimate betas for a matrix dataset*

Description

This function estimates betas (regression coefficients) for fixed and random effects in a matrix dataset using various methods.

Usage

```
## S3 method for class 'matrix_dataset'
estimate_betas(
  x,
  fixed = NULL,
  ran,
  block,
  method = c("mixed", "lss", "ols"),
  basemod = NULL,
  fracs = 0.5,
  progress = TRUE,
  ...
)
```

Arguments

`x`	An object of class `matrix_dataset` representing the matrix dataset
`fixed`	A formula specifying the fixed regressors that model constant effects (i.e., non-varying over trials)
`ran`	A formula specifying the random (trialwise) regressors that model single trial effects

block	A formula specifying the block factor
method	The regression method for estimating trialwise betas; one of "mixed", "lss", or "ols" (default: "mixed")
basemod	A baseline_model instance to regress out of data before beta estimation (default: NULL)
fracs	Fraction of voxels used for prewhitening.
progress	Logical; show progress bar.
...	Additional arguments passed to the estimation method

Value

A list of class "fmri_betas" containing the following components:

- betas_fixed: Matrix representing the fixed effect betas
- betas_ran: Matrix representing the random effect betas
- design_ran: Design matrix for random effects
- design_fixed: Design matrix for fixed effects
- design_base: Design matrix for baseline model

See Also

matrix_dataset, baseline_model

Other estimate_betas: estimate_betas()

---

estimate_hrf *Estimate hemodynamic response function (HRF) using Generalized Additive Models (GAMs)*

Description

This function estimates the HRF using GAMs from the mgcv package. The HRF can be estimated with or without fixed effects.

Usage

```
estimate_hrf(
  form,
  fixed = NULL,
  block,
  dataset,
  bs = c("tp", "ts", "cr", "ps"),
  rsam = seq(0, 20, by = 1),
  basemod = NULL,
  k = 8,
  fx = TRUE,
  progress = TRUE
)
```

Arguments

form	A formula specifying the event model for the conditions of interest
fixed	A formula specifying the fixed regressors that model constant effects (i.e., non-varying over trials); default is NULL
block	A formula specifying the block factor
dataset	An object representing the fMRI dataset
bs	Basis function for the smooth term in the GAM; one of "tp" (default), "ts", "cr", or "ps"
rsam	A sequence of time points at which the HRF is estimated (default: seq(0, 20, by = 1))
basemod	A baseline_model instance to regress out of data before HRF estimation (default: NULL)
k	the dimension of the basis, default is 8
fx	indicates whether the term is a fixed d.f. regression spline (TRUE) or a penalized regression spline (FALSE); default is TRUE.
progress	Logical; display progress during estimation.

Value

A matrix with the estimated HRF values for each voxel

See Also

baseline_model, event_model, design_matrix

Examples

```
# To be added
```

---

evaluate_method_performance

Evaluate Method Performance on Benchmark Dataset

Description

Helper function to evaluate the performance of beta estimation methods on benchmark datasets by comparing estimated betas to ground truth.

Usage

```
evaluate_method_performance(
  dataset_name,
  estimated_betas,
  method_name = "Unknown"
)
```

Arguments

dataset_name	Character string specifying which dataset to use
estimated_betas	Matrix of estimated beta values (conditions x voxels)
method_name	Character string describing the method used

Value

A list with performance metrics

Examples

```
## Not run:
# Load dataset and create design matrix
dataset <- load_benchmark_dataset("BM_Canonical_HighSNR")
X <- create_design_matrix_from_benchmark("BM_Canonical_HighSNR", fmrihrf::HRF_SPMG1)

# Fit simple linear model
betas <- solve(t(X) %*% X) %*% t(X) %*% dataset$Y_noisy

# Evaluate performance
performance <- evaluate_method_performance("BM_Canonical_HighSNR",
                                           betas[-1, ], # Remove intercept
                                           "OLS")

## End(Not run)
```

---

event_table.convolved_term

Extract event table from convolved term

Description

Extract the event table from a convolved term object.

Usage

```
## S3 method for class 'convolved_term'
event_table(x, ...)
```

Arguments

x	A convolved_term object
...	Additional arguments passed to the underlying event_table method

Value

A data.frame containing the event table

extract_csv_data *Extract Data for Meta-Analysis from CSV*

Description

Extract Data for Meta-Analysis from CSV

Usage

```
extract_csv_data(gd, roi = NULL, contrast = NULL)
```

Arguments

gd	A group_data_csv object
roi	Optional ROI name to extract
contrast	Optional contrast name to extract

Value

List with effect sizes and variances

Examples

```
gd <- fmrireg:::.demo_group_data_csv()
extract_csv_data(gd, roi = "ROI1")
```

extract_nuisance_timeseries *Extract Nuisance Timeseries from Mask*

Description

Extracts voxel timeseries from regions defined by a mask (e.g., WM/CSF). This is the typical input for soft subspace projection.

Usage

```
extract_nuisance_timeseries(dataset, mask, run = NULL)
```

Arguments

dataset	An fmri_dataset object.
mask	A binary mask (logical vector or 3D array) indicating nuisance voxels, or a file path to a NIfTI mask.
run	Optional run index to extract data from a specific run.

Value

Matrix of nuisance timeseries (time x nuisance_voxels).

---

fit_contrasts *Fit Contrasts*

Description

Generic function for fitting contrasts to model objects.

Usage

```
fit_contrasts(object, ...)
```

Arguments

object	A fitted model object
...	Additional arguments passed to methods

Value

Contrast results (format depends on method)

---

fit_contrasts.default *Fit Contrasts for Linear Model (Default Method)*

Description

This function calculates contrasts for a fitted linear model.

Usage

```
## Default S3 method:
fit_contrasts(object, conmat, colind, se = TRUE, ...)
```

Arguments

object	The fitted linear model object.
conmat	The contrast matrix or contrast vector.
colind	The subset column indices in the design associated with the contrast.
se	Whether to compute standard errors, t-statistics, and p-values (default: TRUE).
...	Additional arguments (unused)

Value

A list containing the following elements:

- `conmat`: Contrast matrix.
- `sigma`: Residual standard error.
- `df.residual`: Degrees of freedom for residuals.
- `estimate`: Estimated contrasts.
- `se`: Standard errors of the contrasts (if `se = TRUE`).
- `stat`: t-statistics for the contrasts (if `se = TRUE`).
- `prob`: Probabilities associated with the t-statistics (if `se = TRUE`).
- `stat_type`: Type of the statistics calculated.

`fit_contrasts.fmri_lm` *Fit Contrasts for fMRI Linear Model Objects*

Description

S3 method for computing contrasts on fitted fmri_lm objects.

Usage

```
## S3 method for class 'fmri_lm'
fit_contrasts(object, contrasts, ...)
```

Arguments

`object`	An fmri_lm object
`contrasts`	A list of contrast specifications
`...`	Additional arguments (unused)

Value

A list of contrast results

`fit_glm_from_suffstats` *Fit GLM from sufficient statistics*

Description

Computes OLS/GLS-equivalent estimates from cross-products without materializing the transformed series. Engines can stream `XtX`, `XtS`, and `StS` and call this to obtain a standard `fmri_lm` object. AR/robust are not estimated here; this is a low-level helper for OLS-equivalent inference from suffstats.

Usage

```
fit_glm_from_suffstats(
  model,
  XtX,
  XtS,
  StS,
  df,
  cfg = NULL,
  dataset = NULL,
  strategy = "external",
  engine = "external"
)
```

Arguments

`model`	An `fmri_model` describing the design.
`XtX`	p×p cross-product of the design matrix.
`XtS`	p×V cross-product of design with data.
`StS`	length-V vector of sum of squares per voxel.
`df`	Residual degrees of freedom.
`cfg`	Optional `fmri_lm_config`; used for metadata only.
`dataset`	Optional dataset backing the model.
`strategy`	Character label for the returned object.
`engine`	Character label for the returned object.

Value

An object of class `fmri_lm`.

fit_glm_on_transformed_series *Fit the core GLM on a transformed time-series matrix*

Description

Fit the core GLM on a transformed time-series matrix

Usage

```
fit_glm_on_transformed_series(
  model,
  Y,
  cfg = NULL,
  dataset = NULL,
  strategy = "external",
  engine = "external"
)
```

Arguments

model	An fmri_model describing the design.
Y	Numeric matrix with nrow(Y) time points and columns matching voxels.
cfg	Optional fmri_lm_config; defaults to fmri_lm_control().
dataset	Optional dataset backing the model. Defaults to model$dataset when available.
strategy	Character label recorded on the returned object. Defaults to "external".
engine	Character label indicating the engine that produced the fit. Defaults to "external".

Value

An object of class fmri_lm.

fit_glm_with_config *Fit GLM with full config (AR/robust) on a transformed series*

Description

Runs the same integrated solver used by fmri_lm, honoring AR/robust options from cfg, but on an externally provided response matrix Y (T×V). This is intended for engines that transform the time-series before inference.

Usage

```
fit_glm_with_config(
  model,
  Y,
  cfg = NULL,
  dataset = NULL,
  strategy = "external",
  engine = "external"
)
```

Arguments

model	An fmri_model describing the design.
Y	Numeric matrix with nrow(Y) time points and columns matching voxels.
cfg	Optional fmri_lm_config; defaults to fmri_lm_control().
dataset	Optional dataset backing the model. Defaults to model$dataset when available.
strategy	Character label recorded on the returned object. Defaults to "external".
engine	Character label indicating the engine that produced the fit. Defaults to "external".

Value

An object of class fmri_lm.

---

fitted_hrf *fitted_hrf*

Description

Compute and return the fitted hemodynamic response function (HRF) for a model object. The HRF represents the expected BOLD response to neural activity. For models with multiple basis functions, this returns the combined HRF shape.

Usage

```
fitted_hrf(x, sample_at, ...)
```

Arguments

x	An object for which the fitted HRF should be computed
sample_at	A vector of time points at which the HRF should be sampled
...	Additional arguments passed to methods

Details

This generic function computes the fitted hemodynamic response function (HRF) for an object. The method needs to be implemented for specific object types.

Value

A numeric vector containing the fitted HRF values at the requested time points

See Also

fmrihrf::HRF_SPMG1(), fmri_lm()

Examples

```
# Create a simple dataset with two conditions
X <- matrix(rnorm(100 * 100), 100, 100)  # 100 timepoints, 100 voxels
event_data <- data.frame(
  condition = factor(c("A", "B", "A", "B")),
  onsets = c(1, 25, 50, 75),
  run = c(1, 1, 1, 1)
)

# Create dataset and sampling frame
dset <- fmridataset::matrix_dataset(X, TR = 2, run_length = 100, event_table = event_data)
sframe <- sampling_frame(blocklens = 100, TR = 2)

# Create event model with canonical HRF
evmodel <- event_model(
  onsets ~ hrf(condition),
  data = event_data,
  block = ~run,
  sampling_frame = sframe
)

# Fit model
fit <- fmri_lm(
  onsets ~ hrf(condition),
  block = ~run,
  dataset = dset
)

# Get fitted HRF at specific timepoints
times <- seq(0, 20, by = 0.5)  # Sample from 0-20s every 0.5s
hrf_values <- fitted_hrf(fit, sample_at = times)
```

`fitted_hrf.fmri_lm` *Compute Fitted Hemodynamic Response Functions for an fmri_lm Object*

Description

This method computes the fitted hemodynamic response functions (HRFs) for an `fmri_lm` object.

Usage

```
## S3 method for class 'fmri_lm'
fitted_hrf(x, sample_at = seq(0, 24, by = 1), ...)
```

Arguments

`x`	An `fmri_lm` object for which the fitted HRFs should be computed.
`sample_at`	A numeric vector of time points at which the HRFs should be sampled. Default is `seq(0, 24, by = 1)`.
`...`	Additional arguments (currently unused).

Value

A list where each element corresponds to an event term in the `fmri_lm` object. Each element contains:

`pred` A matrix of predicted HRF values.

`design` A tibble containing the design matrix for the HRFs.

`flip_sign` *Flip Sign of Coefficients*

Description

Reverses the sign of coefficient-like outputs in a fit object. Useful for switching between A-B and B-A conventions.

Usage

```
flip_sign(fit, coef = NULL)
```

Arguments

`fit`	An fmri_ttest_fit or similar object
`coef`	Character vector of coefficient names to flip (default: all)

Value

Modified fit object with flipped signs

fmri_benchmark_datasets *Benchmark fMRI datasets*

Description

A list of simulated datasets used for testing the package.

Format

A list with elements `BM_Canonical_HighSNR`, `BM_Canonical_LowSNR`, `BM_HRF_Variability_AcrossVoxels`, `BM_Trial_Amplitude_Variability`, `BM_Complex_Realistic`, and `metadata`. Each element contains simulated BOLD data and ground truth information.

Source

Generated by `data-raw/generate_benchmark_datasets.R`

fmri_latent_lm *Fast fMRI Regression Model Estimation from a Latent Component Dataset*

Description

This function estimates a regression model for fMRI data using a latent component dataset. The dataset must be of type `latent_dataset`, which itself requires a `LatentNeuroVec` input.

Usage

```
fmri_latent_lm(
  formula,
  block,
  baseline_model = NULL,
  dataset,
  durations,
  drop_empty = TRUE,
  robust = FALSE,
  autocor = c("none", "auto", "ar1", "ar2", "arma"),
  bootstrap = FALSE,
  nboot = 1000,
  ...
)
```

Arguments

formula	A formula specifying the regression model.
block	A factor indicating the block structure of the data.
baseline_model	An optional baseline model.
dataset	A dataset of class 'latent_dataset'.
durations	The duration of events in the dataset.
drop_empty	Whether to drop empty events from the model. Default is TRUE.
robust	Whether to use robust regression methods. Default is FALSE.
autocor	The autocorrelation correction method to use on components. One of 'none', 'auto', 'ar1', 'ar2', or 'arma'. Default is 'none'.
bootstrap	Whether to compute bootstrapped parameter estimates. Default is FALSE.
nboot	The number of bootstrap iterations. Default is 1000.
...	Additional arguments.

Value

An object of class 'fmri_latent_lm' containing the regression model and dataset.

Note

This method is currently experimental.

Examples

```
# Estimate the fMRI regression model using the latent dataset
#result <- fmri_latent_lm(formula = formula, block = block, dataset = dset,
#                         durations = NULL, drop_empty = TRUE, robust = FALSE)

# Print the result
#print(result)
```

---

fmri_lm *Fit a Linear Regression Model for fMRI Data Analysis*

Description

fmri_lm is a generic for fitting fMRI regression models. The default interface accepts a model formula and dataset. An alternative method can be used with a preconstructed fmri_model object that already contains the design and data.

Usage

```
fmri_lm(formula, ...)

## S3 method for class 'formula'
fmri_lm(
  formula,
  block,
  baseline_model = NULL,
  dataset,
  durations = 0,
  drop_empty = TRUE,
  robust = FALSE,
  robust_options = NULL,
  ar_options = NULL,
  volume_weights_options = NULL,
  soft_subspace_options = NULL,
  strategy = c("runwise", "chunkwise"),
  nchunks = 10,
  use_fast_path = FALSE,
  progress = FALSE,
  ar_voxelwise = FALSE,
  parallel_voxels = FALSE,
  cor_struct = NULL,
  cor_iter = NULL,
  cor_global = NULL,
  ar1_exact_first = NULL,
  ar_p = NULL,
  robust_psi = NULL,
  robust_max_iter = NULL,
  robust_scale_scope = NULL,
  volume_weights = NULL,
  nuisance_projection = NULL,
  parallel_chunks = FALSE,
  ...
)

## S3 method for class 'fmri_model'
fmri_lm(
  formula,
  dataset = NULL,
  robust = FALSE,
  robust_options = NULL,
  ar_options = NULL,
  volume_weights_options = NULL,
  soft_subspace_options = NULL,
  strategy = c("runwise", "chunkwise"),
  nchunks = 10,
  use_fast_path = FALSE,
```

```
  progress = FALSE,
  ar_voxelwise = FALSE,
  parallel_voxels = FALSE,
  cor_struct = NULL,
  cor_iter = NULL,
  cor_global = NULL,
  ar1_exact_first = NULL,
  ar_p = NULL,
  robust_psi = NULL,
  robust_max_iter = NULL,
  robust_scale_scope = NULL,
  volume_weights = NULL,
  nuisance_projection = NULL,
  parallel_chunks = FALSE,
  ...
)
```

Arguments

formula	A model formula describing the event structure or an fmri_model object.
...	Additional method arguments. Recognized engine arguments include engine, engine_args, and lowrank; see Details.
block	The model formula for block structure.
baseline_model	(Optional) A baseline_model object. Default is NULL.
dataset	An fmri_dataset object containing the time-series data.
durations	A vector of event durations. Default is 0.
drop_empty	Logical. Whether to remove factor levels with zero size. Default is TRUE.
robust	Logical or character. Either FALSE (no robust fitting), TRUE (use Huber), or one of "huber" or "bisquare". Default is FALSE.
robust_options	List of robust fitting options. See Details.
ar_options	List of autoregressive modeling options. See Details.
volume_weights_options	List of volume weighting options. See fmri_lm_control.
soft_subspace_options	List of soft subspace projection options. See fmri_lm_control.
strategy	The data splitting strategy, either "runwise" or "chunkwise". Default is "runwise".
nchunks	Number of data chunks when strategy is "chunkwise". This controls memory partitioning; chunks are processed sequentially unless parallel_chunks = TRUE. Default is 10.
use_fast_path	Logical. If TRUE, use matrix-based computation for speed. Default is FALSE.
progress	Logical. Whether to display a progress bar during model fitting. Default is FALSE.
ar_voxelwise	Logical. Estimate AR parameters voxel-wise (overrides ar_options$voxelwise).

parallel_voxels
: Logical. Parallelize across voxels where supported; this does not control chunk-wise execution.

cor_struct
: Character. Shorthand for ar_options$struct (e.g., "ar1", "ar2", "arp").

cor_iter
: Integer. Shorthand for ar_options$iter_gls.

cor_global
: Logical. Shorthand for ar_options$global.

ar1_exact_first
: Logical. Shorthand for ar_options$exact_first.

ar_p
: Integer. Shorthand for ar_options$p.

robust_psi
: Character. Shorthand for robust_options$type (e.g., "huber", "bisquare").

robust_max_iter
: Integer. Shorthand for robust_options$max_iter.

robust_scale_scope
: Character. Shorthand for robust_options$scale_scope ("run", "global", or "voxel").

volume_weights
: Logical or character. Simple interface for volume weighting:

  - TRUE: Enable with default method ("inverse_squared")
  - "inverse_squared", "soft_threshold", "tukey": Enable with specific method
  - FALSE or NULL: Disable (default)

  For fine-grained control, use volume_weights_options instead.

nuisance_projection
: Matrix, character path, or NULL. Simple interface for soft subspace projection:

  - Matrix: Use as nuisance timeseries directly
  - Character: Path to NIfTI mask for WM/CSF voxels
  - NULL: Disable (default)

  For fine-grained control (lambda selection, warnings), use soft_subspace_options.

parallel_chunks
: Logical. For strategy = "chunkwise", process chunks with future.apply::future_lapply() using the active future plan. Default is FALSE.

Details

robust_options may contain:

- type: Character or logical. Type of robust fitting (FALSE, "huber", "bisquare")
- k_huber: Numeric. Tuning constant for Huber's psi (default: 1.345)
- c_tukey: Numeric. Tuning constant for Tukey's bisquare psi (default: 4.685)
- max_iter: Integer. Maximum IRLS iterations (default: 2)
- scale_scope: Character. Scope for scale estimation ("run" or "global")
- reestimate_phi: Logical. Whether to re-estimate AR parameters after robust fitting

ar_options may contain:

- struct: Character. Correlation structure ("iid", "ar1", "ar2", "arp")
- p: Integer. AR order when struct = "arp"
- iter_gls: Integer. Number of GLS iterations (default: 1)
- global: Logical. Use global AR coefficients (default: FALSE)
- voxelwise: Logical. Estimate AR parameters voxel-wise (default: FALSE)
- exact_first: Logical. Apply exact AR(1) scaling to first sample (default: FALSE)

Built-in fast engines are selected through ...:

- engine = "latent_sketch" uses the sketched GLM path. Configure it with lowrank = lowrank_control(...). The alias engine = "sketch" is normalized to "latent_sketch".
- engine = "rrr_gls" uses the reduced-rank-regression GLS path. Configure it with engine_args = list(...). This engine supports shared AR whitening from ar_options and inference for event/task parameters only; nuisance or baseline terms may be estimated, but standard-error and contrast inference is restricted to event/task coefficients.

For engine = "rrr_gls", engine_args may contain:

- rank: Positive integer rank. If NULL and rank_mode = "fixed", the engine uses the full available task rank.
- rank_mode: One of "fixed", "energy", or "rss_budget" (default: "fixed").
- energy_keep: Fraction of task singular-value energy to retain when rank_mode = "energy" (default: 0.99).
- rss_budget: Non-negative residual-sum-of-squares budget used when rank_mode = "rss_budget".
- se_mode: Standard-error mode, either "conditional" or "bootstrap" (default: "conditional"). Bootstrap standard errors resample task-subspace residuals in blocks.
- bootstrap_n: Number of bootstrap resamples when se_mode = "bootstrap" (default: 200).
- bootstrap_block_size: Positive integer block size for bootstrap resampling. If NULL, a block size of 1 is used.
- bootstrap_seed: Optional integer seed for reproducible bootstrap resampling.
- contrast_policy: How to handle post-hoc contrasts that include non-event coefficients: "warn_drop", "drop", or "error" (default: "warn_drop").

The aliases energy and nboot are accepted as legacy shorthand for energy_keep and bootstrap_n, respectively. Prefer the canonical names in new code.

Value

An object of class fmri_lm.

A fitted linear regression model for fMRI data analysis.

See Also

fmri_dataset, fmri_lm_fit, fmri_lm_control

Examples

```
facedes <- subset(read.table(system.file("extdata", "face_design.txt", package = "fmrireg"),
header=TRUE), face_gen != "n/a")
facedes$face_gen <- droplevels(factor(facedes$face_gen))
sframe <- sampling_frame(rep(430/2,6), TR=2)
ev <- event_model(onset ~ hrf(face_gen, basis="gaussian"), data=facedes,
block= ~ run, sampling_frame=sframe)
globonsets <- fmrihrf::global_onsets(sframe, facedes$onset, facedes$run)
reg1_signal <- regressor(globonsets[facedes$face_gen == "male"], hrf=fmrihrf::HRF_GAUSSIAN)
reg2_signal <- regressor(globonsets[facedes$face_gen == "female"], hrf=fmrihrf::HRF_GAUSSIAN)
time <- samples(sframe, global=TRUE)
y1 <- fmrihrf::evaluate(reg1_signal, time)*1.5
y2 <- fmrihrf::evaluate(reg2_signal, time)*3.0
y <- y1+y2
ys1 <- y + rnorm(length(y), sd=.02)
ys2 <- y + rnorm(length(y), sd=.02)

h <<- gen_hrf(fmrihrf::hrf_bspline, N=7, span=25)
dset <- matrix_dataset(cbind(ys1,ys2), TR=2,
                       run_length=fmrihrf::blocklens(sframe),
                       event_table=facedes)
flm <- fmri_lm(onset ~ hrf(face_gen,
                           basis=gen_hrf(fmrihrf::hrf_bspline, N=7, span=25)),
               block = ~ run,
               strategy="chunkwise", nchunks=1, dataset=dset)

## Not run:
fit_rrr <- fmri_lm(
  onset ~ hrf(condition), block = ~ run, dataset = dset,
  ar_options = list(struct = "ar1"),
  engine = "rrr_gls",
  engine_args = list(
    rank_mode = "energy",
    energy_keep = 0.99,
    se_mode = "bootstrap",
    bootstrap_n = 200L,
    bootstrap_block_size = 24L,
    bootstrap_seed = 1L
  )
)
standard_error(fit_rrr, type = "estimates")

## End(Not run)
```

`fmri_lm_control` *Configuration for fmri_lm fitting*

Description

fmri_lm_control() creates an fmri_lm_config object collecting all options for robust and autoregressive modelling. It validates inputs and applies defaults so downstream functions receive a single structured list.

Usage

```
fmri_lm_control(
  robust_options = list(),
  ar_options = list(),
  volume_weights_options = list(),
  soft_subspace_options = list()
)
```

Arguments

robust_options	list of robust fitting options. See Details.
ar_options	list of autoregressive modelling options. See Details.
volume_weights_options	list of volume weighting options. See Details. For simple cases, use the volume_weights parameter in fmri_lm() instead.
soft_subspace_options	list of soft subspace projection options. See Details. For simple cases, use the nuisance_projection parameter in fmri_lm() instead.

Details

For common use cases, fmri_lm() provides convenience parameters that are easier to use than these detailed option lists:

- volume_weights = TRUE enables volume weighting with defaults
- volume_weights = "tukey" enables with Tukey method
- nuisance_projection = N enables soft projection with matrix N
- nuisance_projection = "mask.nii" enables with mask file

Use the *_options lists below only when you need fine-grained control.

robust_options may contain:

- type (FALSE, "huber", "bisquare")
- k_huber
- c_tukey
- max_iter
- scale_scope ("run", "global", "voxel")
- reestimate_phi (logical)

ar_options may contain:

- `struct` ("iid", "ar1", "ar2", "arp")
- `p` (order for "arp")
- `iter_gls` (integer number of GLS iterations)
- `global` (logical, use global phi)
- `voxelwise` (logical)
- `exact_first` (logical)
- `censor` (integer vector of timepoints to exclude from AR estimation, logical vector where TRUE = censored, or "auto" to extract from dataset)

`volume_weights_options` may contain:

- `enabled` (logical, whether to compute and apply volume weights)
- `method` ("inverse_squared", "soft_threshold", "tukey")
- `threshold` (numeric, DVARS threshold for weighting)
- `weights` (optional pre-computed weight vector)

`soft_subspace_options` may contain:

- `enabled` (logical, whether to apply soft subspace projection)
- `nuisance_mask` (path to NIfTI mask or logical vector)
- `nuisance_matrix` (pre-computed nuisance timeseries matrix)
- `lambda` (numeric, "auto", or "gcv")
- `warn_redundant` (logical, warn if baseline has nuisance terms)

This list controls soft subspace preprocessing. It is separate from the built-in fast `fmri_lm()` engines. To use reduced-rank-regression GLS with conditional or block-bootstrap standard errors, call `fmri_lm()` with `engine = "rrr_gls"` and pass reduced-rank options in `engine_args`. To use the sketched GLM path, call `fmri_lm()` with `engine = "latent_sketch"` and pass a `lowrank_control()` object via `lowrank`.

When `fmri_lm()` is called with the convenience argument `nuisance_projection`, `enabled` is set automatically. When constructing a `soft_subspace_options` list directly, set `enabled = TRUE` yourself.

Value

An object of class `fmri_lm_config`.

`fmri_meta` *Fit Group-Level Meta-Analysis*

Description

Performs voxelwise or ROI-based meta-analysis on group fMRI data using fixed-effects, random-effects, or robust methods. Supports meta-regression with covariates for group comparisons and other moderator analyses.

Usage

```
fmri_meta(
  data,
  formula = ~1,
  method = c("pm", "fe", "dl", "reml"),
  robust = c("none", "huber", "t"),
  weights = c("ivw", "equal", "custom"),
  weights_custom = NULL,
  combine = NULL,
  contrasts = NULL,
  return_cov = NULL,
  chunk_size = 10000,
  n_threads = getOption("fmrireg.num_threads", 0),
  verbose = TRUE
)
```

Arguments

`data`	A group_data object created by `group_data`
`formula`	Formula specifying the meta-regression model. Default is ~ 1 (intercept only). Use ~ 1 + group for group comparisons, or include continuous covariates.
`method`	Character string specifying the meta-analysis method: • "fe": Fixed-effects (inverse variance weighted) • "pm": Paule-Mandel random-effects (default, good for whole-brain) • "dl": DerSimonian-Laird random-effects • "reml": Restricted maximum likelihood random-effects
`robust`	Character string specifying robust estimation: • "none": No robust estimation (default) • "huber": Huber M-estimator with IRLS • "t": Student-t mixture model (for heavy-tailed distributions)
`weights`	Character string specifying weighting scheme: • "ivw": Inverse variance weighting (default) • "equal": Equal weights for all subjects • "custom": User-provided weights (must supply weights argument)

weights_custom	Numeric vector or matrix of custom weights (required if weights = "custom"). If a vector, length must equal the number of subjects. If a matrix, it must be subjects x features.
combine	For t-statistic-only data, combination method ("stouffer", "fisher", or "lancaster"). Stouffer combines z-scores and supports equal, inverse-variance, or custom weighting (via weights). Fisher uses equal weights. Lancaster implements a weighted Fisher method by mapping weights to per-subject degrees of freedom.
contrasts	Optional numeric vector or matrix specifying fit-time exact contrasts. If a vector is provided, its names must match the column names of the design matrix X. A matrix should have columns corresponding to predictors and rows corresponding to contrasts.
return_cov	Optional. If set to "tri", returns the packed upper-triangular Var(beta) per feature under $cov to enable exact post-hoc contrasts via contrast().
chunk_size	Number of voxels to process at once (default: 10000)
n_threads	Number of parallel threads to use. Defaults to fmrireg.num_threads option.
verbose	Logical. Print progress messages (default: TRUE)

Value

An fmri_meta object containing:

- coefficients: Meta-regression coefficients
- se: Standard errors
- tau2: Between-study variance (for random-effects)
- I2: I-squared heterogeneity statistic
- Q: Cochran's Q statistic
- model: Model specification
- data: Input group_data object

Examples

```
## Not run:
# Simple fixed-effects meta-analysis
fit <- fmri_meta(gd, method = "fe")

# Random-effects with group comparison
fit <- fmri_meta(gd, formula = ~ 1 + group, method = "pm")

# Robust meta-regression with continuous covariate
fit <- fmri_meta(gd, formula = ~ 1 + age + sex, method = "reml", robust = "huber")

# Stouffer's Z for t-statistics only
fit <- fmri_meta(gd_tstat, combine = "stouffer")

# Exact post-hoc contrasts by storing covariance
fit_cov <- fmri_meta(gd, formula = ~ 1 + group, method = "pm", return_cov = "tri")
con <- contrast(fit_cov, c("(Intercept)" = 0, group = 1))
```

```
# Exact fit-time contrast without storing covariance
fit_con <- fmri_meta(gd, formula = ~ 1 + group, method = "pm",
                     contrasts = c("(Intercept)" = 0, group = 1))

## End(Not run)
```

`fmri_meta_fit` *Fit Meta-Analysis Models*

Description

Low-level function that calls the C++ meta-analysis implementation. This is typically called internally by higher-level functions like fmri_meta().

Usage

```
fmri_meta_fit(
  Y,
  V,
  X,
  method = c("pm", "dl", "fe", "reml"),
  robust = c("none", "huber"),
  huber_c = 1.345,
  robust_iter = 2,
  n_threads = getOption("fmrireg.num_threads", 0)
)
```

Arguments

`Y`	Numeric matrix of effect sizes (subjects x features)
`V`	Numeric matrix of variances (subjects x features)
`X`	Numeric matrix; design matrix (subjects x predictors), including intercept
`method`	Character scalar; meta-analysis method: "pm" (Paule-Mandel), "dl" (DerSimonian-Laird), "fe" (fixed-effects), or "reml" (REML, uses PM solver)
`robust`	Character scalar; robust estimation method: "none" or "huber"
`huber_c`	Numeric scalar; tuning constant for Huber M-estimator (default: 1.345). Smaller values provide more robust estimates but may reduce efficiency.
`robust_iter`	Integer scalar; number of IRLS iterations for robust estimation (default: 2)
`n_threads`	Integer scalar; number of OpenMP threads (0 = use all available)

Value

List with components:

`beta`	Numeric matrix of coefficients (predictors x features)
`se`	Numeric matrix of standard errors (predictors x features)
`z`	Numeric matrix of z-scores (predictors x features)
`tau2`	Numeric vector of between-study variance estimates
`Q_fe`	Numeric vector of Q statistics from fixed-effects model
`I2_fe`	Numeric vector of I^2 statistics from fixed-effects model
`df`	Numeric vector of degrees of freedom
`ok`	Logical vector indicating successful fits

See Also

`fmri_meta`

`fmri_meta_fit_contrasts` *Fit Meta-Analysis Models with Exact Contrasts*

Description

Low-level function that calls the C++ meta-analysis implementation and returns exact contrast statistics c' (X' W X)^(-1) c for provided contrasts.

Usage

```
fmri_meta_fit_contrasts(
  Y,
  V,
  X,
  Cmat,
  method = c("pm", "dl", "fe", "reml"),
  robust = c("none", "huber"),
  huber_c = 1.345,
  robust_iter = 2,
  n_threads = getOption("fmrireg.num_threads", 0)
)
```

Arguments

Y, V, X	See fmri_meta_fit
Cmat	Numeric matrix K x J (columns are contrasts over predictors)
method	Character scalar; meta-analysis method: "pm" (Paule-Mandel), "dl" (DerSimonian-Laird), "fe" (fixed-effects), or "reml" (REML, uses PM solver)
robust	Character scalar; robust estimation method: "none" or "huber"
huber_c	Numeric scalar; tuning constant for Huber M-estimator (default: 1.345). Smaller values provide more robust estimates but may reduce efficiency.
robust_iter	Integer scalar; number of IRLS iterations for robust estimation (default: 2)
n_threads	Integer scalar; number of OpenMP threads (0 = use all available)

Value

List with base meta outputs plus c_beta, c_se, c_z

---

fmri_meta_fit_cov *Fit Meta-Analysis and return packed covariance per voxel*

Description

Fit Meta-Analysis and return packed covariance per voxel

Usage

```
fmri_meta_fit_cov(
  Y,
  V,
  X,
  method = c("pm", "dl", "fe", "reml"),
  robust = c("none", "huber"),
  huber_c = 1.345,
  robust_iter = 2,
  n_threads = getOption("fmrireg.num_threads", 0)
)
```

Arguments

Y	Numeric matrix of effect sizes (subjects x features)
V	Numeric matrix of variances (subjects x features)
X	Numeric matrix; design matrix (subjects x predictors), including intercept
method	Character scalar; meta-analysis method: "pm" (Paule-Mandel), "dl" (DerSimonian-Laird), "fe" (fixed-effects), or "reml" (REML, uses PM solver)
robust	Character scalar; robust estimation method: "none" or "huber"

huber_c	Numeric scalar; tuning constant for Huber M-estimator (default: 1.345). Smaller values provide more robust estimates but may reduce efficiency.
robust_iter	Integer scalar; number of IRLS iterations for robust estimation (default: 2)
n_threads	Integer scalar; number of OpenMP threads (0 = use all available)

Value

List with base outputs and cov_tri (tsize x P) where tsize = K*(K+1)/2

`fmri_meta_fit_extended` *Extended Meta-Analysis Fit with Voxelwise Covariate*

Description

Wrapper for meta-analysis that supports an optional voxelwise covariate. This extends the basic fmri_meta_fit to handle per-voxel covariates.

Usage

```
fmri_meta_fit_extended(
  Y,
  V,
  X,
  method = c("pm", "dl", "fe", "reml"),
  robust = c("none", "huber"),
  huber_c = 1.345,
  robust_iter = 2,
  voxelwise = NULL,
  center_voxelwise = TRUE,
  voxel_name = "voxel_cov",
  n_threads = getOption("fmrireg.num_threads", 0)
)
```

Arguments

Y	Matrix of effect sizes (S x P)
V	Matrix of variances (S x P)
X	Design matrix (S x K)
method	Meta-analysis method
robust	Robust estimation method
huber_c	Huber tuning constant
robust_iter	Number of IRLS iterations
voxelwise	Optional voxelwise covariate matrix (S x P)

center_voxelwise
: Logical; center voxelwise covariate per feature

voxel_name
: Name for voxelwise coefficient

n_threads
: Number of threads

Value

List with meta-analysis results

`fmri_model` *Construct an fMRI Regression Model*

Description

This function constructs an fMRI regression model consisting of an event model and a baseline model. The resulting model can be used for the analysis of fMRI data.

Usage

```
fmri_model(event_model, baseline_model, dataset)
```

Arguments

`event_model`
: An object of class "event_model" representing the event-related part of the fMRI regression model.

`baseline_model`
: An object of class "baseline_model" representing the baseline-related part of the fMRI regression model.

`dataset`
: An `fmri_dataset` used to build the model.

Value

An object of class `fmri_model` containing the event and baseline models along with the dataset.

See Also

event_model, baseline_model

fmri_ols_fit *OLS Fit with Optional Voxelwise Covariate*

Description

Wrapper for OLS t-tests that supports an optional voxelwise covariate.

Usage

```
fmri_ols_fit(
  Y,
  X,
  voxelwise = NULL,
  center_voxelwise = TRUE,
  voxel_name = "voxel_cov"
)
```

Arguments

Y	Outcome matrix (S x P)
X	Design matrix (S x K)
voxelwise	Optional voxelwise covariate matrix (S x P)
center_voxelwise	Logical; center voxelwise covariate per feature
voxel_name	Name for voxelwise coefficient

Value

List with OLS results

fmri_rlm *Fit a Robust Linear Model for fMRI Data Analysis*

Description

This function fits a robust linear regression model for fMRI data analysis using the specified model formula, block structure, and dataset. The model can be fit using either a runwise or chunkwise data splitting strategy.

Usage

```
fmri_rlm(
  formula,
  block,
  baseline_model = NULL,
  dataset,
  durations = 0,
  drop_empty = TRUE,
  strategy = c("runwise", "chunkwise"),
  nchunks = 10,
  cor_struct = c("iid", "ar1", "ar2", "arp"),
  cor_iter = 1L,
  cor_global = FALSE,
  ar_p = NULL,
  ar1_exact_first = FALSE,
  robust_psi = c("huber", "bisquare"),
  robust_k_huber = 1.345,
  robust_c_tukey = 4.685,
  robust_max_iter = 2L,
  robust_scale_scope = c("run", "global", "voxel"),
  ...
)
```

Arguments

formula	A model formula describing the event structure or an fmri_model object.
block	The model formula for block structure.
baseline_model	(Optional) A baseline_model object. Default is NULL.
dataset	An fmri_dataset object containing the time-series data.
durations	A vector of event durations. Default is 0.
drop_empty	Logical. Whether to remove factor levels with zero size. Default is TRUE.
strategy	The data splitting strategy, either "runwise" or "chunkwise". Default is "runwise".
nchunks	Number of data chunks when strategy is "chunkwise". Default is 10.
cor_struct	Correlation structure: "iid", "ar1", "ar2", or "arp". Default is "iid".
cor_iter	Number of iterations for AR parameter estimation. Default is 1.
cor_global	Whether to use global AR parameters. Default is FALSE.
ar_p	Order of autoregressive model for "arp" structure. Default is NULL.
ar1_exact_first	Whether to use exact AR(1) for first iteration. Default is FALSE.
robust_psi	Robust psi function: "huber" or "bisquare". Default is "huber".
robust_k_huber	Tuning constant for Huber's psi. Default is 1.345.
robust_c_tukey	Tuning constant for Tukey's bisquare. Default is 4.685.
robust_max_iter	Maximum iterations for robust fitting. Default is 2.

robust_scale_scope
: Scope for robust scale estimation: "run", "global", or "voxel". Default is "run".

... : Additional arguments passed to fmri_lm

Value

A fitted robust linear regression model for fMRI data analysis.

Examples

```
etab <- data.frame(onset=c(1,30,15,25), fac=factor(c("A", "B", "A", "B")), run=c(1,1,2,2))
etab2 <- data.frame(onset=c(1,30,65,75), fac=factor(c("A", "B", "A", "B")), run=c(1,1,1,1))
mat <- matrix(rnorm(100*100), 100,100)
dset <- fmridataset::matrix_dataset(mat, TR=1, run_length=c(50,50),event_table=etab)
dset2 <- fmridataset::matrix_dataset(mat, TR=1, run_length=c(100),event_table=etab2)
lm.1 <- fmri_rlm(onset ~ hrf(fac), block= ~ run, dataset=dset)
lm.2 <- fmri_rlm(onset ~ hrf(fac), block= ~ run, dataset=dset2)
```

---

fmri_ttest *fmrireg t-tests for Group Analysis*

Description

Performs group-level t-tests on fMRI data with support for one-sample, two-sample, paired, and ANCOVA designs. Provides both meta-analysis and classical t-test engines, mirroring AFNI 3dttest++ functionality within the fmrireg framework.

Usage

```
fmri_ttest(
  gd,
  formula = ~1,
  engine = c("auto", "meta", "classic", "welch"),
  paired = FALSE,
  mu0 = 0,
  contrast = NULL,
  mc = NULL,
  alpha = 0.05,
  sign = c("AminusB", "BminusA"),
  mask = NULL,
  voxelwise_cov = NULL,
  center_voxelwise = TRUE,
  voxel_name = "voxel_cov",
  weights = c("ivw", "equal", "custom"),
  weights_custom = NULL,
  combine = NULL
)
```

Arguments

gd	A group_data object or data frame with subject data
formula	R formula for between-subjects design. Examples: • ~ 1: One-sample t-test • ~ 1 + group: Two-sample t-test • ~ 1 + group + age: ANCOVA with covariate
engine	Character string; analysis engine to use: • "auto": Automatically select based on data (default) • "meta": Use inverse-variance meta-analysis (requires SE) • "classic": Use OLS/Student t-test • "welch": Use Welch t-test for unequal variances
paired	Logical; if TRUE, perform paired t-test on differences (default: FALSE)
mu0	Numeric scalar; constant to test against for one-sample tests (default: 0)
contrast	Optional named numeric vector for linear combination of coefficients
mc	Character string or NULL; multiple comparisons correction: • NULL: No correction (default) • "bh": Benjamini-Hochberg FDR • "by": Benjamini-Yekutieli FDR • "spatial_fdr": Spatially-aware FDR
alpha	Numeric scalar; FDR level if mc is not NULL (default: 0.05)
sign	Character string; sign convention for group differences: • "AminusB": A - B (default) • "BminusA": B - A
mask	Optional mask object or path
voxelwise_cov	Optional S x P matrix of voxelwise covariates
center_voxelwise	Logical; center voxelwise covariate per feature (default: TRUE)
voxel_name	Character string; name for voxelwise coefficient (default: "voxel_cov")
weights	Character string for meta-engine weighting: "ivw" (default, uses provided SE), "equal" (equal weights), or "custom" (supply `weights_custom`). Ignored for classic/welch engines.
weights_custom	Numeric vector (length S) or matrix (S x P) of custom weights when `weights = "custom"`.
combine	Optional. When using the meta engine with t-only inputs (i.e., per-subject t-statistics and df), specify the t-combination method: "stouffer", "fisher", or "lancaster". Passed through to `fmri_meta()` when delegating to the meta engine on group_data_*.

Value

An fmri_ttest_fit object containing:

- beta: Matrix of coefficients
- se: Matrix of standard errors (if available)
- t: Matrix of t-statistics (classic engine)
- z: Matrix of z-scores
- p: Matrix of p-values
- df: Matrix of degrees of freedom
- q: Matrix of FDR-adjusted p-values (if mc is used)
- z_contrast: Vector of contrast z-scores (if contrast is used)
- p_contrast: Vector of contrast p-values (if contrast is used)

Examples

```
## Not run:
# One-sample t-test
fit <- fmri_ttest(gd, formula = ~ 1)

# Two-sample t-test
fit <- fmri_ttest(gd, formula = ~ 1 + group)

# Paired t-test
fit <- fmri_ttest(gd_diff, formula = ~ 1, paired = TRUE)

# ANCOVA with age covariate
fit <- fmri_ttest(gd, formula = ~ 1 + group + age)

# With spatial FDR correction
fit <- fmri_ttest(gd, formula = ~ 1 + group, mc = "spatial_fdr", alpha = 0.05)

## End(Not run)
```

---

fmrireg_cli *Run the fmrireg command line interface*

Description

Runs the package-owned command line interface for `fmrireg`. The current public command surface focuses on bundled benchmark dataset inspection through the `benchmark` subcommands.

Usage

```
fmrireg_cli(args = commandArgs(trailingOnly = TRUE))
```

Arguments

args	Character vector of command line arguments.

Value

Integer exit status. Returns 0 on success, 1 for command-level validation failures, and 2 for usage or runtime errors.

Examples

```
fmrireg_cli(c("benchmark", "list"))
fmrireg_cli(c(
  "benchmark", "summary",
  "--dataset", "BM_Canonical_HighSNR",
  "--json"
))
```

---

generate_interaction_contrast

Fast factorial contrast generators

Description

Returns a matrix **N_cells x N_contrasts** - *each row is a design cell*, columns are independent contrasts (difference-coded for the factors you ask for, grand-mean for the rest). Suitable for `tcrossprod(dm, C)` or `lm.fit(design, y)` followed by `%*% coef` in the usual way.

Usage

```
generate_interaction_contrast(des, factors)

generate_main_effect_contrast(des, factor)
```

Arguments

des	data.frame with one column per factor (must be `factor`)
factors	character vector: which factor(s) get **difference coding**. - `generate_main_effect_contrast()` takes a **single** factor name. - `generate_interaction_contrast()` takes >= 2 for an interaction (or 1 to reproduce a main-effect matrix).
factor	Single factor name for the main effect.

Value

numeric matrix **nrow = prod levels(f) , ncol = prod (Li - 1)** for the chosen factors.

Examples

```
des <- expand.grid(Time = factor(1:4),
                   Cond = factor(c("face","scene")))

# Main effect of Time (4-1 = 3 contrasts)
M <- generate_main_effect_contrast(des, "Time")

# Full TimexCond interaction ( (4-1)*(2-1) = 3 contrasts )
I <- generate_interaction_contrast(des, c("Time","Cond"))
dim(I)   # 8 rows (cells) x 3 columns (contrasts)
```

---

get_benchmark_summary *Get Benchmark Dataset Summary*

Description

Provides a detailed summary of a specific benchmark dataset including dimensions, experimental design, and ground truth information.

Usage

```
get_benchmark_summary(dataset_name)
```

Arguments

dataset_name	Character string specifying which dataset to summarize

Value

A list with summary information about the dataset

Examples

```
# Get summary of a specific dataset
summary_info <- get_benchmark_summary("BM_Canonical_HighSNR")
print(summary_info)
```

get_contrasts *Get Available Contrasts*

Description

Get Available Contrasts

Usage

```
get_contrasts(gd)
```

Arguments

gd	A group_data_csv object

Value

Character vector of contrast names

Examples

```
gd <- fmrireg:::.demo_group_data_csv()
get_contrasts(gd)
```

get_covariates *Get Covariates*

Description

Get Covariates

Usage

```
get_covariates(x)
```

Arguments

x	A group_data object

Value

Data frame of covariates or NULL

Examples

```
gd <- fmrireg:::.demo_group_data_csv()
get_covariates(gd)
```

get_rois *Get Available ROIs*

Description

Get Available ROIs

Usage

```
get_rois(gd)
```

Arguments

gd	A group_data_csv object

Value

Character vector of ROI names

Examples

```
gd <- fmrireg:::.demo_group_data_csv()
get_rois(gd)
```

get_subjects *Get Subject IDs*

Description

Get Subject IDs

Usage

```
get_subjects(x)
```

Arguments

x	A group_data object

Value

Character vector of unique subject IDs

Examples

```
gd <- fmrireg:::.demo_group_data_csv()
get_subjects(gd)
```

glm_lss *GLM LSS Estimation Convenience Function (Single Trial Estimation)*

Description

A convenience wrapper around `estimate_betas` for least squares separate (LSS) estimation. **This is primarily designed for single trial estimation**, where each individual trial/event gets its own separate beta estimate rather than averaging across trials of the same condition.

Usage

```
glm_lss(
  dataset,
  model_obj,
  basis_obj,
  basemod = NULL,
  block = ~1,
  use_cpp = FALSE,
  progress = TRUE,
  ...
)
```

Arguments

`dataset`	A `matrix_dataset` object containing the fMRI time series data
`model_obj`	An `event_model` object specifying the experimental design
`basis_obj`	An HRF basis object (e.g., from `fmrihrf::HRF_SPMG1`, `HRF_FIR`, etc.)
`basemod`	A `baseline_model` instance to regress out of data before beta estimation (default: NULL)
`block`	A formula specifying the block factor (default: ~ 1 for single block)
`use_cpp`	Deprecated. The C++ implementation has been retired. This parameter is ignored; fmrilss is always used.
`progress`	Logical; show progress bar (default: TRUE)
`...`	Additional arguments passed to `estimate_betas`

Details

Primary Use Case - Single Trial Estimation:

- **Trial-wise beta estimation**: Each trial gets its own beta coefficient
- **Single trial analysis**: Useful for decoding, representational similarity analysis (RSA)
- **Trial-by-trial variability**: Captures individual trial responses rather than condition averages
- **Avoiding trial averaging**: Preserves trial-specific information that would be lost in standard GLM

Method Details: LSS (Least Squares Separate) fits a separate model for each trial, where the trial of interest gets its own regressor while all other trials of the same condition are modeled together. This approach avoids the collinearity issues that would arise from including separate regressors for every trial simultaneously.

For standard condition-level estimation (averaging trials within conditions), use `glm_ols()` instead.

Value

A list of class "fmri_betas" containing the estimated trial-wise coefficients

See Also

`estimate_betas` for the underlying estimation function, `glm_ols` for condition-level estimation

Examples

```
## Not run:
# Create event model and data
event_data <- data.frame(
  onset = c(10, 30, 50, 70),
  condition = factor(c("A", "B", "A", "B")),
  run = rep(1, 4)
)
sframe <- fmrihrf::sampling_frame(blocklens = 100, TR = 2)
model_obj <- event_model(onset ~ hrf(condition),
                         data = event_data,
                         block = ~ run,
                         sampling_frame = sframe)

# Create data matrix (100 timepoints, 10 voxels)
Y <- matrix(rnorm(1000), 100, 10)

# Create matrix_dataset with event table
dset <- matrix_dataset(Y, TR = 2, run_length = 100, event_table = event_data)

# Fit with LSS - estimates separate beta for each individual trial
fit <- glm_lss(dset, model_obj, fmrihrf::HRF_SPMG1)
dim(fit$betas_ran)  # 4 trials x 10 voxels (NOT averaged by condition)

# This is useful for:
# - Decoding analysis (predicting condition from single trial patterns)
# - RSA (representational similarity analysis)
# - Studying trial-by-trial variability

## End(Not run)
```

glm_ols *GLM OLS Estimation Convenience Function*

Description

A convenience wrapper around `estimate_betas` for ordinary least squares (OLS) estimation. This function provides a simplified interface for fitting GLMs using OLS on matrix datasets.

Usage

```
glm_ols(
  dataset,
  model_obj,
  basis_obj,
  basemod = NULL,
  block = ~1,
  progress = TRUE,
  ...
)
```

Arguments

`dataset`	A `matrix_dataset` object containing the fMRI time series data
`model_obj`	An `event_model` object specifying the experimental design
`basis_obj`	An HRF basis object (e.g., from `fmrihrf::HRF_SPMG1`, `HRF_FIR`, etc.)
`basemod`	A `baseline_model` instance to regress out of data before beta estimation (default: NULL)
`block`	A formula specifying the block factor (default: ~ 1 for single block)
`progress`	Logical; show progress bar (default: TRUE)
`...`	Additional arguments passed to `estimate_betas`

Details

Use Cases:

- **Condition-level estimation**: Estimates average responses for each experimental condition
- **General linear modeling**: Standard GLM approach for group-level or condition-level effects
- **Multi-trial averaging**: Combines trials of the same condition to estimate mean responses

For single-trial estimation where each trial gets its own beta estimate, use `glm_lss()` instead.

Value

A list of class "fmri_betas" containing the estimated coefficients

See Also

estimate_betas for the underlying estimation function, glm_lss for single trial estimation

Examples

```
## Not run:
# Create event model and data
event_data <- data.frame(
  onset = c(10, 30, 50, 70),
  condition = factor(c("A", "B", "A", "B")),
  run = rep(1, 4)
)
sframe <- fmrihrf::sampling_frame(blocklens = 100, TR = 2)
model_obj <- event_model(onset ~ hrf(condition),
                         data = event_data,
                         block = ~ run,
                         sampling_frame = sframe)

# Create data matrix (100 timepoints, 10 voxels)
Y <- matrix(rnorm(1000), 100, 10)

# Create matrix_dataset with event table
dset <- matrix_dataset(Y, TR = 2, run_length = 100, event_table = event_data)

# Fit with OLS - estimates average response for each condition
fit <- glm_ols(dset, model_obj, fmrihrf::HRF_SPMG1)
dim(fit$betas_ran)  # 2 conditions x 10 voxels

## End(Not run)
```

---

group_data *Create Group Dataset for Meta-Analysis*

Description

Generic constructor that creates a group dataset from various input formats for use in group-level meta-analysis with fmri_meta.

Usage

```
group_data(data, format = c("auto", "h5", "nifti", "csv", "fmrilm"), ...)
```

Arguments

data	Input data. Format depends on the format argument: • For "h5": Character vector of HDF5 file paths • For "nifti": List or data frame with beta/SE/variance paths

- For "csv": Path to CSV file or data frame
- For "fmrilm": List of fmri_lm objects

format	Character string specifying the input format. One of "auto" (default), "h5", "nifti", "csv", or "fmrilm". If "auto", attempts to detect format.
...	Additional arguments passed to format-specific constructors

Value

A group_data object suitable for meta-analysis

Examples

```
## Not run:
# From HDF5 files created by write_results.fmri_lm
gd <- group_data(
  h5_paths,
  format = "h5",
  subjects = subject_ids,
  covariates = covariates_df,
  contrast = "FaceVsPlace"
)

# From NIfTI files
gd <- group_data(
  list(beta = beta_paths, se = se_paths),
  format = "nifti",
  subjects = subject_ids,
  mask = "group_mask.nii.gz"
)

## End(Not run)
```

---

group_data_from_csv *Create Group Dataset from CSV File or Data Frame*

Description

Creates a group dataset from tabular data containing pre-extracted statistics such as ROI means, effect sizes, and standard errors. This format is useful for ROI-based analyses or when working with summary statistics.

Usage

```
group_data_from_csv(
  data,
  effect_cols,
  subject_col = "subject",
  roi_col = NULL,
```

```
  contrast_col = NULL,
  covariate_cols = NULL,
  wide_format = FALSE
)
```

Arguments

data	Either a path to a CSV file or a data frame containing the data
effect_cols	Named vector or list specifying column names for effect statistics. E.g., c(beta = "mean_activation", se = "std_error") or c(t = "t_stat", df = "df")
subject_col	Character string specifying the column containing subject IDs
roi_col	Character string specifying the column containing ROI names (optional)
contrast_col	Character string specifying the column containing contrast names (optional)
covariate_cols	Character vector of column names to use as covariates (optional)
wide_format	Logical. If TRUE, expects wide format with ROIs as columns (default: FALSE)

Value

A group_data_csv object

Examples

```
## Not run:
# Long format: one row per subject-ROI combination
gd <- group_data_from_csv(
  "roi_statistics.csv",
  effect_cols = c(beta = "mean_beta", se = "se_beta"),
  subject_col = "participant_id",
  roi_col = "roi_name",
  covariate_cols = c("age", "sex", "group")
)

# Wide format: one row per subject, ROIs as columns
gd <- group_data_from_csv(
  "subject_summary.csv",
  effect_cols = c(beta = "roi_"),  # Prefix for ROI columns
  subject_col = "subject",
  wide_format = TRUE
)

# From data frame with multiple contrasts
df <- read.csv("contrast_results.csv")
gd <- group_data_from_csv(
  df,
  effect_cols = c(beta = "estimate", se = "std_error", t = "t_value"),
  subject_col = "subject_id",
  contrast_col = "contrast_name",
  roi_col = "region"
)
```

```
## End(Not run)
```

`group_data_from_fmrilm` *Create Group Dataset from fmri_lm Objects*

Description

Creates a group dataset directly from a list of fitted fmri_lm objects

Usage

```
group_data_from_fmrilm(
  lm_list,
  contrast = NULL,
  stat = c("beta", "se"),
  subjects = NULL,
  covariates = NULL
)
```

Arguments

`lm_list`	List of fmri_lm objects
`contrast`	Character string specifying which contrast to extract
`stat`	Character vector of statistics to extract
`subjects`	Character vector of subject identifiers
`covariates`	Data frame of subject-level covariates

Value

A group_data_h5 object (in-memory variant)

`group_data_from_h5` *Create Group Dataset from HDF5 Files*

Description

Creates a group dataset from HDF5 files produced by `write_results.fmri_lm`. These files use the fmristore LabeledVolumeSet format for efficient storage of multiple statistical maps.

Usage

```
group_data_from_h5(
  paths,
  subjects = NULL,
  covariates = NULL,
  mask = NULL,
  contrast = NULL,
  stat = c("beta", "se"),
  validate = TRUE
)
```

Arguments

paths	Character vector of HDF5 file paths, one per subject
subjects	Character vector of subject identifiers. If NULL, extracted from file paths.
covariates	Data frame of subject-level covariates (optional)
mask	Path to mask file or mask object (optional)
contrast	Character string specifying which contrast to extract (for multi-contrast files)
stat	Character vector of statistics to extract (e.g., c("beta", "se", "tstat"))
validate	Logical. Validate that all files exist and contain expected data (default: TRUE)

Value

A group_data_h5 object

Examples

```
## Not run:
# Read HDF5 files from write_results.fmri_lm
subjects <- data.frame(
  subject = sprintf("sub-%02d", 1:20),
  group = rep(c("young", "old"), each = 10),
  age = c(rnorm(10, 25, 3), rnorm(10, 70, 5))
)

h5_paths <- sprintf("derivatives/sub-%02d_task-nback_desc-GLMstatmap_bold.h5", 1:20)

gd <- group_data_from_h5(
  h5_paths,
  subjects = subjects$subject,
  covariates = subjects[c("group", "age")],
  contrast = "FaceVsPlace",
  stat = c("beta", "se")
)

## End(Not run)
```

`group_data_from_nifti` *Create Group Dataset from NIfTI Files*

Description

Creates a group dataset from NIfTI files containing effect sizes and their standard errors or variances. Supports various input configurations including beta/SE pairs, beta/variance pairs, or t-statistics with degrees of freedom.

Usage

```
group_data_from_nifti(
  beta_paths = NULL,
  se_paths = NULL,
  var_paths = NULL,
  t_paths = NULL,
  df = NULL,
  subjects = NULL,
  covariates = NULL,
  mask = NULL,
  target_space = NULL,
  validate = TRUE
)
```

Arguments

`beta_paths`	Character vector of paths to beta/effect size NIfTI files
`se_paths`	Character vector of paths to standard error NIfTI files
`var_paths`	Character vector of paths to variance NIfTI files (alternative to se_paths)
`t_paths`	Character vector of paths to t-statistic NIfTI files
`df`	Degrees of freedom (scalar or vector). Required if using t_paths.
`subjects`	Character vector of subject identifiers. If NULL, extracted from file paths.
`covariates`	Data frame of subject-level covariates (optional)
`mask`	Path to mask NIfTI file or mask object (optional but recommended)
`target_space`	Path to template NIfTI for spatial alignment checking
`validate`	Logical. Validate that all files exist and have matching dimensions (default: TRUE)

Value

A group_data_nifti object

Examples

```
## Not run:
# From FSL FEAT output (COPE and VARCOPE files)
gd <- group_data_from_nifti(
  beta_paths = Sys.glob("feat_output/sub-*/cope1.nii.gz"),
  var_paths = Sys.glob("feat_output/sub-*/varcope1.nii.gz"),
  subjects = sprintf("sub-%02d", 1:20),
  mask = "group_mask.nii.gz"
)

# From SPM contrast images
gd <- group_data_from_nifti(
  beta_paths = Sys.glob("SPM/sub*/con_0001.nii"),
  se_paths = Sys.glob("SPM/sub*/se_0001.nii"),
  mask = "SPM/mask.nii"
)

# From t-statistics only (for Stouffer's Z combination)
gd <- group_data_from_nifti(
  t_paths = Sys.glob("stats/sub-*/tstat1.nii.gz"),
  df = 100,  # Or vector of per-subject df
  mask = "group_mask.nii.gz"
)

## End(Not run)
```

---

hrf_smoothing_kernel *Compute an HRF smoothing kernel*

Description

This function computes a temporal similarity matrix from a series of hemodynamic response functions.

Usage

```
hrf_smoothing_kernel(
  len,
  TR = 2,
  form = onset ~ trialwise(),
  buffer_scans = 3L,
  normalise = TRUE,
  method = c("gram", "cosine")
)
```

Arguments

len	The number of scans.

TR	The repetition time (default is 2 seconds).
form	the trialwise formula expression, see examples.
buffer_scans	The number of scans to buffer before and after the event.
normalise	Whether to normalise the kernel.
method	The method to use for computing the kernel.

Value

a smoothing matrix

Examples

```
form <- onset ~ trialwise(basis="gaussian")
sk <- hrf_smoothing_kernel(100, TR=1.5, form)
```

---

install_cli *Install the fmrireg command wrapper*

Description

Copies the package CLI wrapper from the installed `exec/` directory into a user-selected destination directory and, on Unix-like systems, marks the copied file as executable.

Usage

```
install_cli(dest_dir = "~/.local/bin", overwrite = FALSE, commands = NULL)
```

Arguments

dest_dir	Destination directory for installed command wrappers.
overwrite	Logical; overwrite an existing installed wrapper.
commands	Character vector of command names to install. Defaults to all available commands.

Value

Character vector of installed paths, returned invisibly.

Examples

```
## Not run:
install_cli("~/.local/bin", overwrite = TRUE)

## End(Not run)
```

list_benchmark_datasets

List Available Benchmark Datasets

Description

Returns a summary of all available benchmark datasets with their descriptions.

Usage

```
list_benchmark_datasets()
```

Value

A data.frame with dataset names and descriptions

Examples

```
# See what benchmark datasets are available
list_benchmark_datasets()
```

load_benchmark_dataset

Load fMRI Benchmark Datasets

Description

This function provides easy access to the benchmark datasets included with the fmrireg package. These datasets are designed for testing HRF fitting, beta estimation, and other fMRI analysis methods.

Usage

```
load_benchmark_dataset(dataset_name = "BM_Canonical_HighSNR")
```

Arguments

`dataset_name` Character string specifying which dataset to load. Options include:

- `"BM_Canonical_HighSNR"`: Canonical HRF with high SNR (3 conditions)
- `"BM_Canonical_LowSNR"`: Canonical HRF with low SNR (3 conditions)
- `"BM_HRF_Variability_AcrossVoxels"`: HRF varies across voxel groups (2 conditions)
- `"BM_Trial_Amplitude_Variability"`: Trial-to-trial amplitude variability (1 condition)

- "BM_Complex_Realistic": Complex scenario with multiple factors (3 conditions)
- "all": Returns all datasets as a list
- "metadata": Returns metadata about the datasets

Value

A list containing the specified benchmark dataset(s) with the following components:

- description: Text description of the dataset
- Y_noisy: Matrix of noisy BOLD time series (time x voxels)
- Y_clean: Matrix of clean BOLD time series (when available)
- X_list_true_hrf: List of design matrices convolved with true HRF
- true_hrf_parameters: Information about the true HRF(s) used
- event_onsets: Vector of event onset times
- condition_labels: Vector of condition labels for each event
- true_betas_condition: Matrix of true condition-level beta values
- true_amplitudes_trial: Matrix of true trial-level amplitudes
- TR: Repetition time
- total_time: Total scan duration
- noise_parameters: Information about noise generation
- simulation_seed: Random seed used for generation
- target_snr: Target signal-to-noise ratio

Examples

```
# Load a specific dataset
high_snr_data <- load_benchmark_dataset("BM_Canonical_HighSNR")

# Get information about all available datasets
metadata <- load_benchmark_dataset("metadata")

# Load all datasets
all_data <- load_benchmark_dataset("all")

# Access the BOLD data
Y <- high_snr_data$Y_noisy

# Get event information
onsets <- high_snr_data$event_onsets
conditions <- high_snr_data$condition_labels
```

`longnames` *Extract Long Names of Variable Levels*

Description

Get the extended names of variable levels, which include the term prefix and any basis function information. Long names provide the complete specification of each condition in the model. For example, if a term has conditions "level1" and "level2" with basis functions "basis1" and "basis2", the long names would be "term#level1:basis1", "term#level1:basis2", "term#level2:basis1", "term#level2:basis2".

Usage

```
longnames(x, ...)

## S3 method for class 'event_term'
longnames(x, ...)

## S3 method for class 'event_seq'
longnames(x, ...)

## S3 method for class 'convolved_term'
longnames(x, ...)

## S3 method for class 'event_model'
longnames(x, ...)
```

Arguments

x	The object to extract names from (typically an event_term, event_model, or convolved_term)
...	Additional arguments passed to methods. Common arguments include: **exclude_basis** Logical; if TRUE, exclude basis function labels from names **drop_empty** Logical; if TRUE, drop empty condition levels

Value

A character vector containing the full condition names with term prefixes and basis functions

See Also

`shortnames()`, `event_model()`, `event_term()`

Examples

```
# Create example data with multiple conditions
event_data <- data.frame(
  condition = factor(c("A", "B", "C", "A", "B", "C")),
  rt = c(0.8, 1.2, 0.9, 1.1, 0.7, 1.3),
  onsets = c(1, 10, 20, 30, 40, 50),
  run = c(1, 1, 1, 1, 1, 1)
)

# Create sampling frame
sframe <- sampling_frame(blocklens = 60, TR = 2)

# Create event model with multiple basis functions
evmodel <- event_model(
  onsets ~ hrf(condition, basis = "fourier", nbasis = 2),
  data = event_data,
  block = ~run,
  sampling_frame = sframe
)

# Get long names including basis functions
lnames <- longnames(evmodel)
# Returns: c("condition#A:basis1", "condition#A:basis2",
#           "condition#B:basis1", "condition#B:basis2",
#           "condition#C:basis1", "condition#C:basis2")

# Create simple event term
eterm <- event_term(
  list(condition = event_data$condition),
  onsets = event_data$onsets,
  blockids = event_data$run
)

# Get long names for term
term_names <- longnames(eterm)
# Returns: c("condition#A", "condition#B", "condition#C")
```

---

lowrank_control *Low-rank / sketch controls for fast GLM*

Description

Control object to enable the optional sketched GLM engine.

Usage

```
lowrank_control(
  parcels = NULL,
  landmarks = NULL,
```

```
  k_neighbors = 16L,
  time_sketch = list(method = "gaussian", m = NULL, iters = 0L),
  ncomp = NULL,
  noise_pcs = 0L
)
```

Arguments

parcels	Optional parceling, e.g., a neuroim2::ClusteredNeuroVol or integer vector (length = number of voxels in mask).
landmarks	Optional integer; number of landmark voxels for optional Nyström extension (NULL = off).
k_neighbors	Integer; k for k-NN in Nyström extension.
time_sketch	List(method = "gaussian" | "countsketch", m = NULL, iters = 0L).
ncomp	Optional integer; number of latent components within parcels (PCA).
noise_pcs	Integer; optional GLMdenoise-style PCs from low-R2 parcels.

Value

A list with class "lowrank_control".

---

meta_effective_n *Compute Effective Sample Size for Meta-Analysis*

Description

Computes the effective sample size based on the heterogeneity estimate. This is useful for understanding the impact of between-study heterogeneity.

Usage

```
meta_effective_n(v, tau2)
```

Arguments

v	Numeric vector of within-study variances
tau2	Numeric scalar; between-study variance (tau-squared)

Value

Numeric scalar; effective sample size

See Also

fmri_meta

Examples

```
meta_effective_n(v = rep(0.05, 3), tau2 = 0.01)
```

`meta_fit_vcov_cpp` *Meta-regression with ONE voxelwise covariate*

Description

Meta-regression with ONE voxelwise covariate

Usage

```
meta_fit_vcov_cpp(
  Y,
  V,
  X,
  C,
  method,
  robust,
  huber_c = 1.345,
  robust_iter = 2L,
  n_threads = 0L
)
```

Arguments

`Y`	S x P matrix of effect sizes
`V`	S x P matrix of variances
`X`	S x K design matrix
`C`	S x P matrix of voxelwise covariates
`method`	Meta-analysis method
`robust`	Robust estimation method
`huber_c`	Huber tuning constant
`robust_iter`	Number of IRLS iterations
`n_threads`	Number of OpenMP threads

Value

List with results

`mixed_solve_cpp` *Mixed Model Solver using Rcpp and roptim*

Description

This function solves a mixed model using Rcpp and roptim for optimization. It estimates variance components in a mixed model, potentially speeding up computations compared to the pure R implementation.

Usage

```
mixed_solve_cpp(
  y,
  Z = NULL,
  K = NULL,
  X = NULL,
  method = "REML",
  bounds = c(1e-09, 1e+09),
  SE = FALSE,
  return_Hinv = FALSE
)
```

Arguments

`y`	Response vector.
`Z`	Design matrix for random effects (default: identity matrix of size n).
`K`	Kinship matrix (default: NULL).
`X`	Design matrix for fixed effects (default: vector of ones).
`method`	Optimization method, either "REML" or "ML" (default: "REML").
`bounds`	Bounds for the optimizer (default: c(1e-9, 1e9)).
`SE`	Logical, whether to return standard errors (default: FALSE).
`return_Hinv`	Logical, whether to return the inverse of H (default: FALSE).

Value

A list containing:

`Vu`	Estimated variance component for random effects.
`Ve`	Estimated variance component for residuals.
`beta`	Estimated fixed effects coefficients.
`u`	Estimated random effects coefficients.
`LL`	Log-likelihood of the model.
`beta.SE`	Standard errors of fixed effects coefficients (if SE = TRUE).
`u.SE`	Standard errors of random effects coefficients (if SE = TRUE).
`Hinv`	Inverse of H (if return_Hinv = TRUE).

Examples

```
## Not run:
# Example usage with random data
set.seed(123)
n <- 100
y <- rnorm(n)
Z <- matrix(rnorm(n * 5), n, 5)
K <- diag(5)
X <- matrix(1, n, 1)
result <- mixed_solve_cpp(y, Z, K, X)

## End(Not run)
```

---

n_subjects *Extract Number of Subjects*

Description

Extract Number of Subjects

Usage

```
n_subjects(x)
```

Arguments

x	A group_data object

Value

Integer number of unique subjects

Examples

```
gd <- fmrireg:::.demo_group_data_csv()
n_subjects(gd)
```

ols_t_cpp *OLS t-test / ANCOVA across features*

Description

OLS t-test / ANCOVA across features

Usage

```
ols_t_cpp(Y, X)
```

Arguments

Y	S x P matrix (subjects x features)
X	S x K design matrix with intercept if desired

Value

List with beta (K x P), se (K x P), t (K x P), df (scalar), ok (P)

ols_t_vcov_cpp *OLS with ONE voxelwise covariate*

Description

OLS with ONE voxelwise covariate

Usage

```
ols_t_vcov_cpp(Y, X, C)
```

Arguments

Y	S x P matrix of outcomes
X	S x K design matrix
C	S x P matrix of voxelwise covariates

Value

List with beta ((K+1) x P), se, t, df

p_values *Extract P-values from a Model Fit*

Description

Extract p-values associated with parameter estimates or test statistics from a fitted model object. This is part of a family of functions for extracting statistical measures.

Usage

```
p_values(x, ...)

## S3 method for class 'fmri_lm'
p_values(x, type = c("estimates", "contrasts"), ...)
```

Arguments

x	The fitted model object
...	Additional arguments passed to methods
type	Character string specifying the type of p-values to extract. Options typically include "estimates" for parameter estimates and "contrasts" for contrast tests. Defaults to "estimates" in most methods.

Value

A tibble or matrix containing p-values

See Also

Other statistical_measures: coef_names(), standard_error(), stats()

Examples

```
# Create example data
event_data <- data.frame(
  condition = factor(c("A", "B", "A", "B")),
  onsets = c(1, 10, 20, 30),
  run = c(1, 1, 1, 1)
)

# Create sampling frame and dataset
sframe <- sampling_frame(blocklens = 50, TR = 2)
dset <- fmridataset::matrix_dataset(
  matrix(rnorm(50 * 2), 50, 2),
  TR = 2,
  run_length = 50,
  event_table = event_data
)
```

```
# Fit model
fit <- fmri_lm(
  onsets ~ hrf(condition),
  block = ~run,
  dataset = dset
)

# Extract p-values
pvals <- p_values(fit)
```

`paired_diff_block` *Helper Functions for fmri_ttest*

Description

Support functions for paired differences, sign flipping, and wrapper functions for OLS and meta-analysis with voxelwise covariates. Compute Paired Within-Subject Differences

Usage

```
paired_diff_block(blkA, blkB, rho = 0)
```

Arguments

`blkA`	First group_data block
`blkB`	Second group_data block
`rho`	Optional within-subject correlation between A and B. Can be: • Scalar: Same correlation for all subjects and features • Vector of length S: Per-subject correlations • Vector of length P: Per-feature correlations • Matrix (S x P): Subject-feature specific correlations Default is 0 (independence).

Details

Creates within-subject differences (A - B) for paired t-tests from two blocks with identical subjects and features.

Value

A new block with Y = Y_A - Y_B and propagated variance if available

print.fmri_meta *Print Meta-Analysis Results*

Description

Print Meta-Analysis Results

Usage

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

Arguments

x	An fmri_meta object
...	Additional print arguments

Value

Invisibly returns the input object x

Examples

```
toy_meta <- structure(
  list(
    coefficients = matrix(0.2, nrow = 1),
    se = matrix(0.05, nrow = 1),
    method = "DL",
    robust = "none",
    formula = ~ condition,
    n_subjects = 12,
    n_rois = 1
  ),
  class = "fmri_meta"
)
print(toy_meta)
```

print.fmri_model *Print an fmri_lm_result object*

Description

Provides a colorful and informative printout.

Usage

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

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

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

Arguments

x	An fmri_rlm object
...	Additional arguments passed to print.fmri_lm

Value

Invisibly returns the input object x

---

print.fmri_ttest_fit *Print method for fmri_ttest_fit*

Description

Print method for fmri_ttest_fit

Usage

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

Arguments

x	An fmri_ttest_fit object
...	Additional print arguments

Value

Invisibly returns the input object x

print.group_data *Print Group Data Object*

Description

Print Group Data Object

Usage

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

Arguments

x	A group_data object
...	Additional print arguments

Value

Invisibly returns the input object x

print.spatial_fdr_result *Print Spatial FDR Results*

Description

Print Spatial FDR Results

Usage

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

Arguments

x	A spatial_fdr_result object
...	Additional print arguments

Value

Invisibly returns the input object x

pvalues *Compute P-values from Meta-Analysis*

Description

Compute P-values from Meta-Analysis

Usage

```
pvalues(object, two_tailed = TRUE)
```

Arguments

object	An fmri_meta object
two_tailed	Logical. Use two-tailed test (default: TRUE)

Value

Matrix of p-values

Examples

```
toy_meta <- structure(
  list(
    coefficients = matrix(c(0.2, -0.1), nrow = 1),
    se = matrix(c(0.05, 0.08), nrow = 1)
  ),
  class = "fmri_meta"
)
pvalues(toy_meta)
```

r_to_z *Convert Correlation to Fisher's Z*

Description

Transforms correlations to Fisher's Z scale for meta-analysis.

Usage

```
r_to_z(r, n)
```

Arguments

r	Numeric vector or matrix of correlations
n	Integer scalar; sample size

Value

List with components:

z	Numeric vector or matrix; Fisher's Z transformed correlations
v	Numeric vector or matrix; sampling variances

See Also

fmri_meta

Examples

```
r_to_z(r = 0.4, n = 30)
```

`read_h5_full` *Read All Data from HDF5 Files*

Description

Reads complete data from all subjects' HDF5 files. Warning: This can use a lot of memory for whole-brain data.

Usage

```
read_h5_full(gd, stat = NULL)
```

Arguments

gd	A group_data_h5 object
stat	Character vector of statistics to extract

Value

Array with dimensions (voxels, subjects, statistics)

read_nifti_full *Read All Data from NIfTI Files*

Description

Reads complete data from all subjects' NIfTI files using memory mapping when possible.

Usage

```
read_nifti_full(gd, use_mask = NULL)
```

Arguments

gd	A group_data_nifti object
use_mask	Logical. Apply mask to data (default: TRUE if mask exists)

Value

List with data matrices

register_basis *Register an HRF/basis constructor*

Description

Register an HRF/basis constructor

Usage

```
register_basis(name, constructor)
```

Arguments

name	Character scalar used in formulas (e.g. `basis = "friman2"`).
constructor	Function returning an object understood by `fmrihrf` (typically an `HRF`). Additional arguments from the original `hrf()` call are forwarded when the basis is constructed.

Value

Invisibly, `TRUE`.

register_engine *Register a plugin engine for* fmri_lm

Description

Register a plugin engine for fmri_lm

Usage

```
register_engine(name, fit, preflight = NULL, capabilities = list())
```

Arguments

name	Character scalar identifier advertised to users (e.g. "friman").
fit	Function invoked as fit(model, dataset, args, cfg) and expected to return an fmri_lm object.
preflight	Optional function invoked before fitting; receives the same arguments as fit and can signal errors early.
capabilities	Optional named list describing engine support for global fmri_lm() options. Recognized fields currently include robust, preprocessing, ar_voxelwise, ar_by_cluster, plus contextual rules such as requires_event_regressors, requires_parcels_for_by_cluster, and forbid_by_cluster_dataset_classes.

Value

Invisibly, TRUE.

resolve_basis *Resolve a registered basis function by name*

Description

This function is used internally by the formula processing system to lazily resolve basis names (e.g., "cca2", "cca3", "spmg1") into actual basis objects. It's part of the plugin API that allows extension packages to register custom basis functions.

Usage

```
resolve_basis(name, ...)
```

Arguments

name	Character string naming a registered basis function
...	Additional arguments passed to the basis constructor

Value

An HRF basis object

Examples

```
## Not run:
# Resolve the cca3 basis with specific parameters
basis <- resolve_basis("cca3", span = 30, TR = 2)

## End(Not run)
```

---

se *Extract Standard Errors from Meta-Analysis*

Description

Extract Standard Errors from Meta-Analysis

Usage

```
se(object)
```

Arguments

`object`	An fmri_meta object

Value

Matrix of standard errors

Examples

```
toy_meta <- structure(
  list(se = matrix(c(0.05, 0.06), nrow = 1)),
  class = "fmri_meta"
)
se(toy_meta)
```

shortnames *Short Names*

Description

Generate short names for model terms and conditions.

Usage

```
shortnames(x, ...)

## S3 method for class 'event_term'
shortnames(x, ...)

## S3 method for class 'event_model'
shortnames(x, ...)
```

Arguments

x	The object to generate short names for.
...	Additional arguments.

Value

A character vector of short names.

simulate_bold_signal *Simulate fMRI Time Series*

Description

This function simulates an fMRI time series for multiple experimental conditions with specified parameters. It generates a realistic event-related design with randomized inter-stimulus intervals and condition orders.

Usage

```
simulate_bold_signal(
  ncond,
  hrf = fmrihrf::HRF_SPMG1,
  nreps = 12,
  amps = rep(1, ncond),
  isi = c(3, 6),
  ampsd = 0,
  TR = 1.5
)
```

Arguments

ncond	The number of conditions to simulate.
hrf	The hemodynamic response function to use (default is fmrihrf::HRF_SPMG1).
nreps	The number of repetitions per condition (default is 12).
amps	A vector of amplitudes for each condition (default is a vector of 1s with length ncond).
isi	A vector of length 2 specifying the range of inter-stimulus intervals to sample from (default is c(3, 6) seconds).
ampsd	The standard deviation of the amplitudes (default is 0).
TR	The repetition time of the fMRI acquisition (default is 1.5 seconds).

Value

A list with the following components:

- onset: A vector of the onset times for each trial
- condition: A vector of condition labels for each trial
- mat: A matrix containing the simulated fMRI time series:
  - Column 1: Time points (in seconds)
  - Columns 2:(ncond+1): Simulated BOLD responses for each condition

Examples

```
# Simulate 3 conditions with different amplitudes
sim <- simulate_bold_signal(ncond = 3, amps = c(1, 1.5, 2), TR = 2)

# Plot the simulated time series
matplot(sim$mat[,1], sim$mat[,-1], type = "l",
        xlab = "Time (s)", ylab = "BOLD Response")
```

---

simulate_fmri_matrix *Simulate fMRI Time Courses, Return Shared Onsets + Column-Specific Amplitudes/Durations*

Description

Generates n time-series (columns) with a single set of onsets, but *resampled* amplitudes/durations for each column if `amplitude_sd>0` or `duration_sd>0`. Each column also gets independent noise. The result is a list containing:

- `time_series`: a `matrix_dataset` with $T \times n$. The `event_table` uses the first column's amplitude/duration draws.
- `ampmat`: an $n\_events \times n$ matrix of per-column amplitudes.
- `durmat`: an $n\_events \times n$ matrix of per-column durations.
- `hrf_info`: info about the HRF.
- `noise_params`: info about noise generation (type + AR coefficients + SD).

Usage

```
simulate_fmri_matrix(
  n = 1,
  total_time = 240,
  TR = 2,
  hrf = fmrihrf::HRF_SPMG1,
  n_events = 10,
  onsets = NULL,
  isi_dist = c("even", "uniform", "exponential"),
  isi_min = 2,
  isi_max = 6,
  isi_rate = 0.25,
  durations = 0,
  duration_sd = 0,
  duration_dist = c("lognormal", "gamma"),
  amplitudes = 1,
  amplitude_sd = 0,
  amplitude_dist = c("lognormal", "gamma", "gaussian"),
  single_trial = FALSE,
  noise_type = c("none", "white", "ar1", "ar2"),
  noise_ar = NULL,
  noise_sd = 1,
  random_seed = NULL,
  verbose = FALSE,
  buffer = 16
)
```

Arguments

n	Number of time-series (columns).
total_time	Numeric. Total scan length (seconds).
TR	Numeric. Repetition time (seconds).
hrf	Hemodynamic response function, e.g. fmrihrf::HRF_SPMG1.
n_events	Number of events (ignored if onsets is provided).
onsets	Optional numeric vector of event onsets. If NULL, will be generated.
isi_dist	One of "even", "uniform", or "exponential". Default is "even" so events are evenly spaced from 0..total_time.
isi_min, isi_max	For isi_dist="uniform".
isi_rate	For isi_dist="exponential".
durations	Numeric, scalar or length-n_events. If duration_sd>0, random sampling is done per column.
duration_sd	Numeric. If >0, random variation in durations.
duration_dist	"lognormal" or "gamma" (strictly positive).

amplitudes	Numeric, scalar or length-n_events. If amplitude_sd>0, random sampling is done per column.
amplitude_sd	Numeric. If >0, random variation in amplitudes.
amplitude_dist	"lognormal", "gamma", or "gaussian" (can be negative).
single_trial	If TRUE, each event is a separate single-trial regressor that gets summed.
noise_type	"none", "white", "ar1", or "ar2".
noise_ar	Numeric vector for AR(1) or AR(2). If missing or insufficient, defaults are used (0.3 for AR(1); c(0.3,0.2) for AR(2)).
noise_sd	Std dev of the noise.
random_seed	Optional integer for reproducibility.
verbose	If TRUE, prints messages.
buffer	Numeric seconds appended to the end of the time grid to avoid edge truncation (default: 16).

Details

- If noise_type="ar1" and you do not provide noise_ar, we default to c(0.3).
- If noise_type="ar2" and you do not provide a 2-element noise_ar, we default to c(0.3, 0.2).
- Onsets are either provided or generated once for all columns.
- **Amplitudes/durations** are re-sampled *inside the loop* so each column can differ randomly. The final arrays ampmat and durmat each have one column per time-series.
- The matrix_dataset's event_table records the first column's amplitudes/durations. If you need each column's, see ampmat and durmat.

Value

A list containing:

time_series A matrix_dataset with $T \times n$ data and event_table for the *first* column's random draws.

ampmat An $n\_events \times n$ numeric matrix of amplitudes.

durmat An $n\_events \times n$ numeric matrix of durations.

hrf_info A list with HRF metadata.

noise_params A list describing noise generation.

`simulate_noise_vector` *Simulate fMRI Noise*

Description

This function simulates realistic fMRI noise by combining:

- Temporal autocorrelation using an ARMA model
- Low-frequency drift
- Physiological noise (cardiac and respiratory)

Usage

```
simulate_noise_vector(
  n,
  TR = 1.5,
  ar = c(0.3),
  ma = c(0.5),
  sd = 1,
  drift_freq = 1/128,
  drift_amplitude = 2,
  physio = TRUE,
  seed = NULL
)
```

Arguments

`n`	The number of time points in the fMRI time series
`TR`	The repetition time in seconds (default is 1.5)
`ar`	A numeric vector containing autoregressive (AR) coefficients (default is c(0.3))
`ma`	A numeric vector containing moving average (MA) coefficients (default is c(0.5))
`sd`	The standard deviation of the white noise component (default is 1)
`drift_freq`	Frequency of the low-frequency drift in Hz (default is 1/128)
`drift_amplitude`	Amplitude of the low-frequency drift (default is 2)
`physio`	Logical; whether to add simulated physiological noise (default is TRUE)
`seed`	An optional seed for reproducibility (default is NULL)

Value

A numeric vector containing the simulated fMRI noise

Examples

```
# Simulate noise for a 5-minute scan with TR=2s
n_timepoints <- 150  # 5 minutes * 60 seconds / 2s TR
noise <- simulate_noise_vector(n_timepoints, TR = 2)
plot(noise, type = "l", xlab = "Time Point", ylab = "Signal")
```

---

simulate_simple_dataset

Simulate Complete fMRI Dataset

Description

This function simulates a complete fMRI dataset by combining task-related signals with realistic noise. It returns both the clean signals and the noisy data.

Usage

```
simulate_simple_dataset(
  ncond,
  nreps = 12,
  TR = 1.5,
  snr = 0.5,
  hrf = fmrihrf::HRF_SPMG1,
  seed = NULL
)
```

Arguments

ncond	Number of conditions to simulate
nreps	Number of repetitions per condition (default is 12)
TR	Repetition time in seconds (default is 1.5)
snr	Signal-to-noise ratio (default is 0.5)
hrf	Hemodynamic response function to use (default is fmrihrf::HRF_SPMG1)
seed	Optional seed for reproducibility (default is NULL)

Value

A list containing:

- clean: The simulated signals without noise (from simulate_bold_signal)
- noisy: The signals with added noise
- noise: The simulated noise component
- onsets: Trial onset times
- conditions: Condition labels for each trial

Examples

```
# Simulate a dataset with 3 conditions
data <- simulate_simple_dataset(ncond = 3, TR = 2, snr = 0.5)

# Plot clean and noisy data
par(mfrow = c(2,1))
matplot(data$clean$mat[,1], data$clean$mat[,-1], type = "l",
        main = "Clean Signal", xlab = "Time (s)", ylab = "BOLD")
matplot(data$noisy[,1], data$noisy[,-1], type = "l",
        main = "Noisy Signal", xlab = "Time (s)", ylab = "BOLD")
```

---

soft_projection *Create Soft Projection Operator*

Description

Computes the soft (ridge-regularized) projection matrix that removes variance explainable by the nuisance subspace while avoiding overfitting.

Usage

```
soft_projection(N, lambda = "auto", Y = NULL)
```

Arguments

N	Nuisance matrix (time x nuisance_voxels). Typically extracted from white matter and CSF voxels. Can have many columns (thousands); computational cost depends on min(nrow, ncol) due to SVD.
lambda	Ridge penalty controlling shrinkage strength: **"auto"** (Default) Uses median of squared singular values. Fast, stable, requires no tuning. Recommended for most use cases. **"gcv"** Generalized cross-validation. Optimizes prediction of Y from N with ridge penalty. Requires Y parameter. More principled but slower. **numeric** User-specified value. Larger = less aggressive removal.
Y	Optional data matrix for GCV-based lambda selection. Required if `lambda = "gcv"`, ignored otherwise.

Value

A list with class `"soft_projection"` containing:

P_lambda	Function that applies the projection to a matrix
lambda	The selected/specified lambda value
method	Method used: "singular_value_heuristic", "gcv", or "user_specified"
effective_df	Effective degrees of freedom removed (sum of shrinkage factors)
n_nuisance	Number of nuisance columns in N
n_timepoints	Number of timepoints

Examples

```
# Create nuisance matrix (e.g., from WM/CSF voxels)
set.seed(123)
N <- matrix(rnorm(100 * 20), nrow = 100, ncol = 20)
proj <- soft_projection(N, lambda = "auto")

# Apply to data and design
Y <- matrix(rnorm(100 * 50), nrow = 100, ncol = 50)
Y_clean <- proj$P_lambda(Y)

# Full workflow: project both data and design
X <- cbind(1, rnorm(100), rnorm(100))  # intercept + 2 predictors
cleaned <- apply_soft_projection(proj, Y, X)
# Now fit GLM with cleaned$Y and cleaned$X

# Using GCV for lambda selection (data-driven)
proj_gcv <- soft_projection(N, lambda = "gcv", Y = Y)
print(proj_gcv)  # Shows selected lambda and effective df
```

---

soft_subspace_options *Soft Subspace Control Options*

Description

Creates a configuration object for soft subspace projection.

Usage

```
soft_subspace_options(
  enabled = FALSE,
  nuisance_mask = NULL,
  nuisance_matrix = NULL,
  lambda = "auto",
  warn_redundant = TRUE
)
```

Arguments

enabled	Logical. Whether to apply soft subspace projection.
nuisance_mask	Path to NIfTI mask or logical vector indicating nuisance voxels.
nuisance_matrix	Pre-computed nuisance timeseries matrix (alternative to mask).
lambda	Ridge penalty: numeric, "auto", or "gcv".
warn_redundant	Logical. Warn if baseline model contains nuisance terms.

Value

A list of class "soft_subspace_options".

Examples

```
# Using a mask file
opts <- soft_subspace_options(
  enabled = TRUE,
  nuisance_mask = "path/to/wm_csf_mask.nii.gz",
  lambda = "auto"
)

# Using pre-computed nuisance matrix
N <- matrix(rnorm(100 * 20), 100, 20)
opts <- soft_subspace_options(
  enabled = TRUE,
  nuisance_matrix = N,
  lambda = 0.5
)
```

---

soft_subspace_projection

Soft Subspace Projection for Nuisance Removal

Description

Ridge-regularized projection to remove nuisance variance without explicitly choosing components or adding confound regressors. This is "CompCor without choosing components."

Conceptual Overview

Traditional CompCor extracts K principal components from white matter/CSF and regresses them out. This requires choosing K, which is arbitrary.

Soft subspace projection instead treats the entire WM/CSF timeseries as a nuisance basis and removes it with shrinkage. Each nuisance direction is partially removed proportional to its variance, avoiding the hard keep/delete decision of PCA truncation.

The Projection Operator

Given nuisance matrix N (time x nuisance_voxels), the soft projection is:

$$P_\lambda = I - N(N^T N + \lambda I)^{-1} N^T$$

Applied to data Y and design X:

- `Y_clean` = `P_lambda %*% Y`
- `X_clean` = `P_lambda %*% X` (important to avoid bias)

Lambda Selection

The shrinkage parameter lambda controls aggressiveness:

- Small lambda: aggressive removal (risk removing signal)
- Large lambda: gentle removal (risk leaving nuisance)

Three selection methods are available:

"auto" Singular value heuristic: `lambda = median(d^2)` where d are singular values of N. Fast, stable, no tuning. Components with variance below median are heavily shrunk; above median lightly shrunk.

"gcv" Generalized cross-validation minimizes `RSS/(1-df/n)^2` for ridge regression of Y on N. Finds lambda giving best leave-one-out prediction without actually doing LOO. Requires Y. Computational cost is O(k) per evaluation where k = min(n, p).

numeric User-specified lambda value.

Typical Usage

The soft subspace projection workflow has three steps:

1. Extract nuisance timeseries from WM/CSF mask (or provide pre-computed)
2. Create the soft projection operator
3. Apply to both data Y and design matrix X before GLM fitting

For use within `fmri_lm()`, see the convenience parameter `nuisance_projection` or the more detailed `soft_subspace_options`.

When to Use

Soft subspace projection is most beneficial when:

- You have physiological noise from WM/CSF that isn't captured by motion parameters
- Traditional CompCor requires arbitrary component selection
- You want to avoid adding many confound regressors to the design

Consider alternatives when:

- Motion parameters alone sufficiently control artifacts
- You prefer explicit confound regressors for interpretability
- Data has very few timepoints relative to nuisance dimensions

spatial_fdr *Spatially-Aware Multiple Comparisons Correction*

Description

Performs spatially-aware FDR control using structure-adaptive weighted Benjamini-Hochberg (SABHA-style) procedure. This method leverages spatial structure in the data to increase power while controlling the false discovery rate.

Usage

```
spatial_fdr(
  z = NULL,
  p = NULL,
  group = NULL,
  alpha = 0.05,
  tau = 0.5,
  lambda = 1,
  neighbors = NULL,
  min_pi0 = 0.05,
  empirical_null = TRUE,
  verbose = FALSE
)
```

Arguments

z	Numeric vector of Z-values (one per feature). Provide either z or p, not both.
p	Numeric vector of p-values (two-sided). Provide either z or p, not both.
group	Integer or factor vector of group IDs for each feature (e.g., parcel IDs, block IDs). Must have same length as z or p.
alpha	Numeric scalar; FDR level to control (default: 0.05)
tau	Numeric scalar; Storey threshold in (0,1) for π_0 estimation (default: 0.5). Higher values are more conservative.
lambda	Numeric scalar; smoothing strength for groupwise π_0 across neighbors (default: 1.0). Set to 0 for no smoothing, higher values for more smoothing.
neighbors	Optional list of length G (number of groups) where each element is an integer vector of 1-based neighbor IDs. Used for spatial smoothing of π_0.
min_pi0	Numeric scalar; lower bound for π_0 to stabilize weights (default: 0.05). Prevents infinite weights.
empirical_null	Logical; if TRUE, estimate null distribution parameters (μ_0, σ_0) from central z-values using robust estimators (default: TRUE).
verbose	Logical; print progress messages (default: FALSE)

Details

This function implements a spatially-aware multiple testing procedure that:

1. Estimates the proportion of null hypotheses (pi0) within spatial groups
2. Optionally smooths these estimates across neighboring groups
3. Uses the pi0 estimates to weight the Benjamini-Hochberg procedure
4. Provides more power in regions with true signal while maintaining FDR control

The method is particularly effective for:

- Voxelwise analyses with spatial clustering of signal
- Parcel-based analyses with anatomical or functional grouping
- Any scenario where hypotheses can be grouped spatially or functionally

Value

Object of class "spatial_fdr_result" containing:

`reject`	Logical vector indicating rejected hypotheses (discoveries)
`q`	Numeric vector of FDR-adjusted p-values (q-values)
`p`	Numeric vector of two-sided p-values used for testing
`weights`	Numeric vector of normalized weights used in weighted BH
`pi0_raw`	Numeric vector of raw π_0 estimates per group
`pi0_smooth`	Numeric vector of smoothed π_0 estimates per group
`threshold`	Numeric scalar; BH threshold used for rejection
`k`	Integer scalar; number of rejections
`mu0`	Numeric scalar; estimated null mean (if empirical_null = TRUE)
`sigma0`	Numeric scalar; estimated null SD (if empirical_null = TRUE)
`groups`	Integer vector of compressed group IDs (1..G)
`group`	Factor or integer vector of original group IDs
`G`	Integer scalar; number of groups
`alpha`	Numeric scalar; FDR level used
`coef_name`	Character scalar; name of coefficient (for S3 method)

References

- Benjamini & Hochberg (1995). Controlling the false discovery rate.
- Storey (2002). A direct approach to false discovery rates.
- Hu et al. (2010). False discovery rate control with groups (SABHA).

See Also

`create_3d_blocks`

Examples

```
# Simple synthetic example
set.seed(123)
n <- 1000
z_scores <- c(rnorm(800), rnorm(200, mean = 2))  # 200 true signals
group_ids <- rep(1:10, each = 100)  # 10 groups of 100 features each

# Basic usage without spatial smoothing
result <- spatial_fdr(z = z_scores, group = group_ids, alpha = 0.05)
summary(result)

# Create simple neighbor structure (each group neighbors with adjacent groups)
neighbors <- lapply(1:10, function(i) {
  c(if(i > 1) i-1, if(i < 10) i+1)
})

# With spatial smoothing
result_smooth <- spatial_fdr(z = z_scores, group = group_ids,
                             neighbors = neighbors, lambda = 1.0)
print(result_smooth)
```

---

standard_error *Extract Standard Errors from a Model Fit*

Description

Extract standard errors of parameter estimates from a fitted model object. This is part of a family of functions for extracting statistical measures.

Usage

```
standard_error(x, ...)

## S3 method for class 'fmri_latent_lm'
standard_error(x, type = c("estimates", "contrasts"), recon = FALSE, ...)

## S3 method for class 'fmri_lm'
standard_error(x, type = c("estimates", "contrasts"), ...)

## S3 method for class 'fmri_lm'
standard_error(x, type = c("estimates", "contrasts"), ...)
```

Arguments

x	The fitted model object
...	Additional arguments passed to methods

type	The type of standard errors to extract: "estimates" or "contrasts" (default: "estimates")
recon	Logical; whether to reconstruct the full matrix representation (default: FALSE)

Value

A tibble or matrix containing standard errors of parameter estimates

See Also

Other statistical_measures: coef_names(), p_values(), stats()

Examples

```
# Create example data
event_data <- data.frame(
  condition = factor(c("A", "B", "A", "B")),
  onsets = c(1, 10, 20, 30),
  run = c(1, 1, 1, 1)
)

# Create sampling frame and dataset
sframe <- sampling_frame(blocklens = 50, TR = 2)
dset <- fmridataset::matrix_dataset(
  matrix(rnorm(50 * 2), 50, 2),
  TR = 2,
  run_length = 50,
  event_table = event_data
)

# Fit model
fit <- fmri_lm(
  onsets ~ hrf(condition),
  block = ~run,
  dataset = dset
)

# Extract standard errors
se <- standard_error(fit)
```

---

stats *Extract Test Statistics from a Model Fit*

Description

Extract test statistics (e.g., t-statistics, F-statistics) from a fitted model object. This is part of a family of functions for extracting statistical measures.

Usage

```
stats(x, ...)

## S3 method for class 'fmri_lm'
stats(x, type = c("estimates", "contrasts", "F"), ...)

## S3 method for class 'fmri_lm'
stats(x, type = c("estimates", "contrasts", "F"), ...)
```

Arguments

x	The fitted model object
...	Additional arguments passed to methods
type	The type of statistics to extract: "estimates", "contrasts", or "F" (default: "estimates")

Value

A tibble or matrix containing test statistics

See Also

Other statistical_measures: coef_names(), p_values(), standard_error()

Examples

```
# Create example data
event_data <- data.frame(
  condition = factor(c("A", "B", "A", "B")),
  onsets = c(1, 10, 20, 30),
  run = c(1, 1, 1, 1)
)

# Create sampling frame and dataset
sframe <- sampling_frame(blocklens = 50, TR = 2)
dset <- fmridataset::matrix_dataset(
  matrix(rnorm(50 * 2), 50, 2),
  TR = 2,
  run_length = 50,
  event_table = event_data
)

# Fit model
fit <- fmri_lm(
  onsets ~ hrf(condition),
  block = ~run,
  dataset = dset
)

# Extract test statistics
tstats <- stats(fit)
```

summary.fmri_meta *Summary of Meta-Analysis Results*

Description

Summary of Meta-Analysis Results

Usage

```
## S3 method for class 'fmri_meta'
summary(object, threshold = 0.05, ...)
```

Arguments

object	An fmri_meta object
threshold	P-value threshold for significance (default: 0.05)
...	Additional summary arguments

Value

A list containing summary statistics invisibly

Examples

```
toy_meta <- structure(
  list(
    coefficients = matrix(c(0.3, -0.1), nrow = 1,
      dimnames = list(NULL, c("A", "B"))),
    se = matrix(c(0.05, 0.07), nrow = 1),
    method = "DL",
    robust = "none",
    formula = ~ condition,
    n_subjects = 12,
    n_rois = 1
  ),
  class = "fmri_meta"
)
summary(toy_meta, threshold = 0.1)
```

summary.fmri_ttest_fit *Summary method for fmri_ttest_fit*

Description

Summary method for fmri_ttest_fit

Usage

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

Arguments

object	An fmri_ttest_fit object
...	Additional summary arguments

Value

Invisibly returns the input object

summary.group_data *Summary of Group Data Object*

Description

Summary of Group Data Object

Usage

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

Arguments

object	A group_data object
...	Additional summary arguments

Value

Invisibly returns the input object

summary.spatial_fdr_result *Summary of Spatial FDR Results*

Description

Summary of Spatial FDR Results

Usage

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

Arguments

object	A spatial_fdr_result object
...	Additional summary arguments

Value

Invisibly returns the input object

t_to_beta_se *Convert t-statistics to Effect Sizes*

Description

Helper function to convert t-statistics and df to beta and SE estimates

Usage

```
t_to_beta_se(t, df, n = NULL)
```

Arguments

t	T-statistic values
df	Degrees of freedom
n	Sample size (optional, improves SE estimation)

Value

List with beta and se estimates

t_to_d *Convert T-statistics to Effect Sizes and Variances*

Description

Converts t-statistics and degrees of freedom to standardized mean differences (Cohen's d) and their sampling variances for meta-analysis.

Usage

```
t_to_d(t, df, n = NULL)
```

Arguments

t	Numeric vector or matrix of t-statistics
df	Numeric scalar or vector; degrees of freedom (matching t)
n	Numeric scalar; sample size per group (for two-sample t-tests)

Value

List with components:

d	Numeric vector or matrix; standardized mean differences
v	Numeric vector or matrix; sampling variances

See Also

fmri_meta

Examples

```
t_to_d(t = 2, df = 18)
t_to_d(t = 2, df = 18, n = 20)
```

term_matrices.fmri_model *Extract term matrices from fmri_model*

Description

Extract design matrices for individual terms from an fmri_model object.

Usage

```
## S3 method for class 'fmri_model'
term_matrices(x, blocknum = NULL, ...)
```

Arguments

x	An fmri_model object
blocknum	Optional vector of block numbers to extract matrices for
...	Additional arguments (currently unused)

Value

A list of matrices, one for each term in the model

Examples

```
fm <- fmrireg:::.demo_fmri_model()
term_matrices(fm)
```

---

tidy *Tidy generic*

Description

Minimal tidy generic to support fmri_meta tidy() without requiring broom.

Usage

```
tidy(x, ...)
```

Arguments

x	object
...	passed to methods

Value

A tidy data frame with model coefficients and statistics

tidy.fmri_meta *Tidy Meta-Analysis Results*

Description

Tidy Meta-Analysis Results

Usage

```
## S3 method for class 'fmri_meta'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

Arguments

x	An fmri_meta object
conf.int	Logical. Include confidence intervals (default: FALSE)
conf.level	Confidence level (default: 0.95)
...	Additional arguments

Value

A tibble with tidy results

tidy_fitted_hrf *Tidy fitted HRF curves from an fmri_lm fit*

Description

Convert `fitted_hrf()` output into a long tibble suitable for plotting.

Usage

```
tidy_fitted_hrf(
  x,
  sample_at = seq(0, 24, by = 1),
  term = NULL,
  term_match = c("contains", "exact", "regex"),
  voxel = 1L,
  average_voxels = FALSE,
  ...
)
```

Arguments

x	An fmri_lm object.
sample_at	Numeric vector of time points passed to fitted_hrf().
term	Optional term selector. When NULL, all event terms are returned.
term_match	Matching mode for term: "contains" (default), "exact", or "regex".
voxel	Integer voxel index to extract from each term's prediction matrix.
average_voxels	Logical; if TRUE, average across voxels instead of selecting one voxel.
...	Additional arguments passed to fitted_hrf().

Value

A tibble with columns term, time, condition, estimate, and voxel.

---

volume_quality *Volume Quality Metrics and Temporal Weighting*

Description

Functions for computing volume-level quality metrics (DVARS) and converting them to weights for weighted least squares fitting. This implements "soft scrubbing" - downweighting bad volumes rather than hard censoring them.

Conceptual Overview

Traditional fMRI artifact removal uses hard censoring ("scrubbing"), where volumes exceeding a threshold are completely removed. This loses temporal information and can create discontinuities.

Soft scrubbing instead assigns each volume a weight between 0 and 1 based on its quality. High-quality volumes receive full weight; artifacts receive reduced weight. This preserves temporal continuity while still downweighting problematic data.

DVARS as Quality Metric

DVARS (Derivative of VARiance across voxels) measures the root mean square of the temporal derivative across all voxels:

$$DVARS_t = \sqrt{\frac{1}{V}\sum_v (Y_{t,v} - Y_{t-1,v})^2}$$

Interpretation:

- Low DVARS: signal changed smoothly from previous volume (good)
- High DVARS: signal changed rapidly (possible artifact or motion)
- Normalized DVARS: values ~1 are typical; >1.5 suggests artifacts

Weighting Methods

Three methods convert DVARS to weights, offering different trade-offs:

inverse_squared Formula: `w = 1 / (1 + dvars^2)`

```
Properties: Smooth, continuous decay. A volume with DVARS=1 gets
weight 0.5. Most conservative choice - provides gentle downweighting
even for moderately elevated DVARS.

Use when: You want smooth, gradual artifact handling without
sharp transitions.
```

soft_threshold Formula: Sigmoid decay above threshold

```
Properties: Volumes below threshold get full weight; above
threshold, weights decay smoothly. Steepness parameter controls
how rapidly weights drop.

Use when: You have a clear idea of what "acceptable" DVARS looks
like (e.g., 1.5x median) and want to preserve good volumes fully.
```

tukey Formula: Tukey bisquare `(1 - u^2)^2` for `|u| <= 1`

```
Properties: Complete downweighting for extreme values. Volumes
beyond 2x threshold get zero weight. Most aggressive choice.

Use when: You have clear artifacts that should be fully excluded,
similar to hard scrubbing but with smooth transitions.
```

When to Use Volume Weighting vs Robust Fitting

Both approaches handle artifacts, but through different mechanisms:

Volume weighting downweights entire timepoints uniformly across all voxels. Best when artifacts affect the whole brain (head motion, scanner spikes).

Robust fitting (Huber/Tukey bisquare) downweights outlier residuals voxel-by-voxel. Best when artifacts are spatially localized or when temporal structure matters.

Combined approach: Use volume weighting for global artifacts plus robust fitting for residual voxel-level outliers. Set `robust = TRUE, volume_weights = TRUE` in `fmri_lm()`.

Typical Usage

For use within `fmri_lm()`, see the convenience parameter `volume_weights` or the more detailed `volume_weights_options`.

volume_weights *Compute Volume Quality Weights from Data*

Description

Convenience function that computes DVARS and converts to weights in one step. This is the main user-facing function for volume quality weighting.

Usage

```
volume_weights(
  Y,
  method = "inverse_squared",
  threshold = 1.5,
  return_dvars = FALSE
)
```

Arguments

Y	Numeric matrix of fMRI data (time x voxels).
method	Weighting method passed to dvars_to_weights.
threshold	Threshold passed to dvars_to_weights.
return_dvars	Logical. If TRUE, return both weights and DVARS.

Value

If return_dvars is FALSE, a numeric vector of weights. If TRUE, a list with components "weights" and "dvars".

Examples

```
set.seed(123)
Y <- matrix(rnorm(100 * 50), nrow = 100, ncol = 50)
Y[50, ] <- Y[50, ] + 5  # Add artifact

# One-step computation of weights
result <- volume_weights(Y, return_dvars = TRUE)
cat("DVARS at artifact:", round(result$dvars[50], 2), "\n")
cat("Weight at artifact:", round(result$weights[50], 3), "\n")

# With Tukey method for more aggressive downweighting
w_tukey <- volume_weights(Y, method = "tukey")
```

welch_t_cpp *Welch two-sample t-test across features*

Description

Welch two-sample t-test across features

Usage

```
welch_t_cpp(Y, g_in)
```

Arguments

Y	S x P matrix
g_in	Length S vector of group indicators (1/2 or 0/1)

Value

List with muA, muB, t, df (Welch), nA, nB

write_results *Write Results from fMRI Analysis*

Description

Generic function to export statistical maps and analysis results from fitted fMRI models to standardized file formats with appropriate metadata.

Usage

```
write_results(x, ...)
```

Arguments

x	A fitted fMRI model object
...	Additional arguments passed to methods

Value

Invisible list of created file paths

`write_results.fmri_lm` *Write Results from fMRI Linear Model*

Description

Exports statistical maps from an fmri_lm object with BIDS-compliant naming and JSON metadata sidecars. Supported image outputs are HDF5 LabeledVolumeSet files and NIfTI files; an fmrigds representation can also be requested.

Usage

```
## S3 method for class 'fmri_lm'
write_results(
  x,
  path = NULL,
  subject = NULL,
  task = NULL,
  space = NULL,
  desc = "GLM",
  format = c("h5"),
  strategy = c("by_stat", "by_contrast"),
  save_betas = TRUE,
  contrasts = NULL,
  contrast_match = c("auto", "exact", "regex"),
  contrast_stats = c("beta", "tstat", "pval", "se"),
  overwrite = FALSE,
  validate_inputs = TRUE,
  ...
)
```

Arguments

`x`	An fmri_lm object containing fitted model results
`path`	Output directory path. If NULL, uses current working directory
`subject`	Subject identifier (e.g., "01", "1001"). Required.
`task`	Task identifier (e.g., "nback", "rest"). Required for BIDS compliance.
`space`	Spatial reference (e.g., "MNI152NLin2009cAsym"). Optional but recommended.
`desc`	Description of the analysis (default: "GLM")
`format`	Output format(s). Use `"h5"` for BIDS-style HDF5 outputs, `"nifti"` for BIDS-style NIfTI outputs, `"gds"` for fmrigds-compatible assays plus an `.rds` plan, or a character vector to write multiple formats.
`strategy`	Storage strategy: "by_stat" (group contrasts by statistic) or "by_contrast" (separate files)
`save_betas`	Logical. Save raw regressor betas (default: TRUE)

contrasts	Character vector of contrast names to save. NULL saves all contrasts
contrast_match	How to match contrasts: "auto" first matches exact contrast names and treats unmatched selectors as regular expressions; "exact" uses literal names only; "regex" treats all selectors as regular expressions.
contrast_stats	Character vector of contrast statistics to save (default: c("beta", "tstat", "pval", "se"))
overwrite	Logical. Overwrite existing files (default: FALSE)
validate_inputs	Logical. Validate fmrilm object structure (default: TRUE)
...	Additional arguments passed to internal functions

Value

Invisible list of file paths created

Examples

```
## Not run:
# Save all results using default settings
write_results(fitted_model, subject = "01", task = "nback")

# Save only specific contrasts and statistics
write_results(fitted_model,
              subject = "01", task = "nback", space = "MNI152NLin2009cAsym",
              contrasts = c("FacesVsPlaces", "GoVsNoGo"),
              contrast_stats = c("beta", "tstat"))

## End(Not run)
```

---

write_results.fmri_meta

Write Meta-Analysis Results

Description

Exports spatial meta-analysis maps with the same core export contract used by write_results.fmri_lm: BIDS-style filenames, HDF5 and NIfTI image outputs, JSON metadata sidecars, vectorized format, explicit overwrite handling, and atomic finalization.

Usage

```
## S3 method for class 'fmri_meta'
write_results(
  x,
  path = NULL,
  subject = NULL,
  task = NULL,
```

```
  space = NULL,
  desc = "Meta",
  format = c("h5"),
  strategy = c("by_stat", "by_coefficient"),
  coefficients = NULL,
  coefficient_match = c("auto", "exact", "regex"),
  coefficient_stats = c("beta", "se", "z", "pval"),
  heterogeneity = TRUE,
  overwrite = FALSE,
  validate_inputs = TRUE,
  ...
)
```

Arguments

x	An fmri_meta object
path	Output directory path. If NULL, uses current working directory.
subject	Optional subject or group identifier. Unlike subject-level GLM exports, meta-analysis outputs may omit this for group-level maps.
task	Optional task identifier.
space	Optional spatial reference label.
desc	Description of the analysis (default: "Meta").
format	Output format(s). Use "h5" for HDF5 LabeledVolumeSet outputs, "nifti" for NIfTI outputs, or a character vector to write both.
strategy	Storage strategy. "by_stat" writes one file per statistic with coefficients along the 4th dimension; "by_coefficient" writes one file per coefficient with statistics along the 4th dimension.
coefficients	Character vector of coefficient names to save. NULL saves all coefficients.
coefficient_match	How to match coefficients: "auto" first matches exact names and treats unmatched selectors as regular expressions; "exact" uses literal names only; "regex" treats all selectors as regular expressions.
coefficient_stats	Character vector of coefficient statistics to save. Supported values are "beta", "se", "z", and "pval".
heterogeneity	Logical. Save heterogeneity maps (tau2, I2, Q, Q_df) when available.
overwrite	Logical. Overwrite existing files (default: FALSE).
validate_inputs	Logical. Validate fmri_meta object structure (default: TRUE).
...	Additional arguments passed to internal functions.

Value

Invisible list of created files

z_to_r *Back-transform Fisher's Z to Correlation*

Description

Back-transform Fisher's Z to Correlation

Usage

```
z_to_r(z)
```

Arguments

z	Numeric vector or matrix; Fisher's Z values

Value

Numeric vector or matrix; correlations

See Also

fmri_meta

Examples

```
z_to_r(0.2)
```

zscores *Compute Z-scores from Meta-Analysis*

Description

Compute Z-scores from Meta-Analysis

Usage

```
zscores(object)
```

Arguments

object	An fmri_meta object

Value

Matrix of z-scores

Examples

```
toy_meta <- structure(
  list(
    coefficients = matrix(c(0.2, -0.1), nrow = 1),
    se = matrix(c(0.05, 0.08), nrow = 1)
  ),
  class = "fmri_meta"
)
zscores(toy_meta)
```

Index