

Package: fmrireg (via r-universe)

October 11, 2026

Type Package

Title Regression Analysis of Functional Magnetic Resonance Imaging Data

Version 0.3.0

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 (>= 4.1.0)

Imports fmriAR (>= 0.3.3), fmrigrds, fmrdesign (>= 0.6.1.9000), fmrihrf (>= 0.4.0), fmrilss (>= 0.1.0), stats, splines, Matrix, MASS, RANN, assertthat, tibble, purrr, dplyr, ggplot2 (>= 3.4.0), tidyr, robustbase, matrixStats, furrr, rlang (>= 1.0.3), neuroim2, Rcpp, RcppParallel, methods, fmrdataset (>= 0.11.0.9000), 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, hdf5r, knitr, rmarkdown, albersdown, roxygen2, covr, gridExtra, multivarious, glmnet, proxy, forecast, future, shiny, sandwich, metafor, progressr, pkgload, roxyglobals (>= 0.2.1), withr, bidser

Remotes bbuchsbaum/bidser@e6d09f85a206dc59db23a086c875b869b5d51891, bbuchsbaum/fmriAR@ca27f2772bb4675573fae717a992c993c2f986c8, bbuchsbaum/fmrihrf@b9f6a90b05bdf3673ed43414eb06ec6c74200d43, bbuchsbaum/fmristore@e7fbbfca181ebd118acee150774a4f58640e039d,

bbuchsbaum/fmrigds@f015b7c7fa5008e2435d269aeb1882a9aa24eaf0,
 anthonymnorth/roxyglobals@ab29bdcd2e618fbfd91356fc22095e961b48333d,
 bbuchsbaum/fmridataset@419d9eae1704d839e0152ec77957c2fc69ea0b5b,
 bbuchsbaum/fmrilss@f851fb0632f90da12512212af5b80c9623c93fde,
 bbuchsbaum/fmrdesign@005606d72d1a5492292d51d3d7b755b3434e2f1c

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 zlib1g-dev

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

Date/Publication 2026-10-11 23:45:06 UTC

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

RemoteRef HEAD

RemoteSha aa36776a8ad3c14ab0ca7555328b957d9502b59d

Contents

.fmrireg_engine_registry	6
.resample_param	6
apply_soft_projection	7
ar_parameters	7
as.array.NeuroVec	8
as_manifest	9
autoplot.fmri_lm	9
autoplot.Reg	12
baseline_spec	13
batch_errors	14
batch_values	14
build_model	15
coef.fmri_lm	15
coef.fmri_meta	17
coef_image.fmri_lm	17
coef_image.fmri_ttest_fit	18
coef_images.fmri_lm	19
coef_names	20
collect_results	21
compute_dvars	22
compute_lm_contrasts	23
compute_lm_contrasts_from_suffstats	24
compute_spec	26
contrast	26
contrast.fmri_meta	27

contrast_set	28
correlation_map.fmri_model	28
create_3d_blocks	29
create_design_matrix_from_benchmark	30
dataset_spec	31
design_map.fmri_model	32
design_matrix.fmri_lm	33
design_matrix.fmri_model	33
design_plot	34
dvars_to_weights	36
engine_spec	37
engine_specs	38
estimate	38
estimate_betas	39
estimate_hrf	42
estimation_spec	44
evaluate_method_performance	44
export_jobs	45
extract_csv_data	46
extract_nuisance_timeseries	47
fit_contrasts	47
fit_contrasts.default	48
fit_contrasts.fmri_lm	49
fit_glm_from_suffstats	49
fit_glm_on_transformed_series	50
fit_glm_with_config	51
fitted_hrf	52
fitted_hrf.fmri_lm	53
flip_sign	54
fmri_benchmark_datasets	54
fmri_hrf_estimate-methods	55
fmri_job	56
fmri_latent_lm	57
fmri_lm	58
fmri_lm_control	59
fmri_meta	60
fmri_meta_fit	63
fmri_meta_fit_contrasts	64
fmri_meta_fit_cov	65
fmri_meta_fit_extended	66
fmri_model	67
fmri_ols_fit	67
fmri_rlm	68
fmri_template	70
fmri_ttest	71
fmrireg_cli	73
from_bids	74
generate_interaction_contrast	75

get_benchmark_summary	76
get_contrasts	77
get_covariates	77
get_rois	78
get_subjects	78
glm_lass	79
glm_ols	81
group_data	82
group_data_from_csv	83
group_data_from_fmriilm	85
group_data_from_h5	86
group_data_from_nifti	87
hrf_smoothing_kernel	88
install_cli	89
instantiate	90
latent_frame	91
list_benchmark_datasets	92
load_benchmark_dataset	92
lowrank_control	94
ma_parameters	95
matrix_frame	96
meta_effective_n	97
meta_fit_vcov_cpp	98
mixed_solve_cpp	99
n_subjects	100
neurovec_frame	101
nifti_frame	102
noise_spec	103
ols_t_cpp	104
ols_t_vcov_cpp	105
p_values	105
paired_diff_block	106
plot_fmri_model	107
preflight	108
print_fmri_meta	109
print_fmri_model	110
print_fmri_ttest_fit	110
print_group_data	111
print_spatial_fdr_result	111
projection_spec	112
pvalues	112
r_to_z	113
read_h5_full	114
read_jobs	114
read_nifti_full	115
realize_dataset	115
reduce_betas	116
reduce_contrasts	116

reduce_identity	117
reduce_write_results	117
register_basis	118
register_engine	119
register_run_backend	119
resolve_basis	120
robust_spec	121
run_backends	121
run_job	122
run_jobs	122
se	123
simulate_bold_signal	124
simulate_fmri_matrix	125
simulate_noise_vector	127
simulate_simple_dataset	128
soft_projection	130
soft_subspace_options	131
soft_subspace_projection	132
spatial_fdr	134
standard_error	136
stats	138
summary.fmri_meta	139
summary.fmri_ttest_fit	140
summary.group_data	141
summary.spatial_fdr_result	141
t_to_beta_se	142
t_to_d	142
term_matrices.fmri_model	143
theme_fmrireg	144
tidy	145
tidy.fmri_hrf_estimate	145
tidy.fmri_meta	146
tidy_fitted_hrf	146
validate_job	147
validate_template	148
variance_model	148
variance_spec	149
volume_quality	149
volume_weights	151
weights_spec	152
welch_t_cpp	152
write_results	153
write_results.fmri_lm	153
write_results.fmri_meta	155
z_to_r	156
zscores	157

```
.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 <code>fmri_lm</code>
...	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	<i>Coerce NeuroVec to base array</i>
-------------------	--------------------------------------

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

as_manifest

*Coerce bindings to an fmri_manifest***Description**

Wraps a list of bindings (or a manifest data.frame) into an fmri_manifest for [instantiate\(\)](#). This is the generic, BIDS-free way to describe subjects; see [from_bids\(\)](#) for the BIDS populator.

Usage

```
as_manifest(x)
```

Arguments

x A list of bindings (named lists, each with at least id, scans, TR, run_length) or a manifest data.frame.

Value

An object of class fmri_manifest.

See Also

[from_bids\(\)](#), [instantiate\(\)](#)

Examples

```
m <- as_manifest(list(
  list(id = "sub-01", scans = matrix(rnorm(80 * 2), 80, 2), TR = 2,
    run_length = c(40, 40),
    events = data.frame(onset = c(5, 45),
      condition = factor(c("A", "B")), run = c(1, 2))))))
length(m)
```

autoplot.fmri_lm

*Plot a fitted fMRI linear model***Description**

Visual summaries of an fmri_lm fit. Five views are available:

Usage

```
## S3 method for class 'fmri_lm'
autoplot(
  object,
  type = c("betas", "contrasts", "hrf", "timecourse", "residuals"),
  voxel = NULL,
  level = 0.95,
  sample_at = seq(0, 20, by = 0.25),
  direct_labels = TRUE,
  ...
)

## S3 method for class 'fmri_lm'
plot(
  x,
  type = c("betas", "contrasts", "hrf", "timecourse", "residuals"),
  voxel = NULL,
  ...
)
```

Arguments

object, x	An fmri_lm object.
type	One of "betas" (default), "contrasts", "hrf", "timecourse", "residuals". The first two name the parameter family, as in <code>stats()</code> and <code>coef()</code> . "estimates" is accepted as a synonym of "betas".
voxel	Optional integer voxel index (or indices, for "betas", "contrasts", "hrf" and "residuals").
level	Confidence level for intervals and bands.
sample_at	Time points (s) at which to evaluate the HRF, for type = "hrf".
direct_labels	Logical; for type = "hrf", label each curve next to it (default) or, if FALSE, use a legend. A legend reads better in small or many-panel figures.
...	Passed on from plot() to autoplot().

Details

"betas" With no voxel, the distribution of t-statistics across voxels for each event regressor, drawn as a sina plot (points spread in proportion to their density), with reference lines at the two-sided $p < .001$ (uncorrected) critical value and the share of voxels beyond it in each tail. With voxel, a coefficient plot of estimates with confidence intervals, one panel per event term so that regressors in different units (a condition and a parametric modulator) do not share an axis.

"contrasts" As "betas", for the fitted contrasts.

"hrf" The estimated hemodynamic response for each condition: the term's HRF basis weighted by the fitted coefficients. For one voxel, the band is a pointwise confidence interval from the coefficient covariance, and a curve whose interval includes zero at its peak is drawn faded and dashed and labelled "n.s.". For several voxels, the line is the mean across voxels, the band a

t-based confidence interval of that mean (voxels treated as independent), and up to 200 voxels are drawn faintly underneath. Peak latency is reported only for terms with more than one basis function; with a single basis the shape, and so the latency, is fixed by the HRF rather than estimated.

"timecourse" Observed signal and model fit for one voxel, with event onsets in a lane per condition and residuals in a lower panel drawn on the same vertical scale.

"residuals" Autocorrelation of the residuals at lags 1 to 10 scans. By default this summarises up to 200 voxels spread evenly through the data (median and 10th-90th percentile per lag); with voxel, it shows those voxels. Least-squares residuals are negatively autocorrelated even when the noise is white, so the 95% white-noise range is centred on the autocorrelation this design induces, not on zero. For fits with an AR noise model the residuals are also shown after prewhitening with the fitted AR coefficients, which is the check that the noise model removed the serial correlation. The subtitle reports a portmanteau test of lags 1-10 that accounts for both the design-induced mean and the correlation between lags that regression introduces (a generalised Ljung-Box test).

When voxel is NULL for the "hrf" and "timecourse" views, the voxel with the largest absolute event t-statistic is shown and named in the subtitle.

Critical values and intervals use the degrees of freedom the fit used for inference (`fit$result$df$inference`), which differ from the nominal residual df for AR, robust, or effective-df corrected fits.

In the "timecourse" and "residuals" views the fitted values are an ordinary least squares fit of the full design (events plus baseline) to the voxel: run by run for runwise fits, jointly otherwise. For OLS fits these are the model's own fitted values; for AR or robust fits they are the unweighted counterpart, on the scale of the data. For runwise fits the fitted values use run-specific coefficients, whereas the estimates view shows coefficients pooled across runs. The reported partial R-squared is $1 - RSS_{full}/RSS_{baseline}$: the share of the variance left by the baseline model that the event regressors explain.

Value

A ggplot2 object. `plot()` prints it and returns it invisibly.

Examples

```
set.seed(1)
ev <- data.frame(onset = seq(10, 180, by = 12), run = 1,
                 condition = factor(rep(c("a", "b"), length.out = 15)))
Y <- matrix(rnorm(100 * 5), 100, 5)
dset <- matrix_frame(Y, TR = 2, run_length = 100, event_table = ev)
fit <- fmri_lm(onset ~ hrf(condition), block = ~ run, dataset = dset)
ggplot2::autoplot(fit)
ggplot2::autoplot(fit, type = "timecourse", voxel = 1)
```

autoplot.Reg

Autoplot method for Reg objects

Description

Plots the predicted BOLD time course of an fMRI regressor: the event train convolved with its HRF. Event onsets are marked as ticks beneath the curve, so the hemodynamic lag is visible. When responses to nearby events overlap, each event's own response is drawn faintly under the summed curve. Multi-basis HRFs are drawn as stacked panels that share the time axis, each with its own vertical scale.

Usage

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

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 spans the first onset to the end of the last event's response (including its duration).
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.
title	Optional plot title, e.g. the condition this regressor models.
...	Additional arguments (currently unused).

Value

A ggplot object.

baseline_spec	<i>Baseline specification (subject-invariant)</i>
---------------	---

Description

Describes how the nuisance / baseline model should be built for every subject, without binding any particular subject's confound values. The concrete `baseline_model` is assembled per subject at instantiation time from this spec, the subject's `sampling_frame`, and the subject's confound matrix.

Usage

```
baseline_spec(
  degree = 3,
  basis = c("bs", "poly", "ns", "constant"),
  confounds = NULL,
  intercept = c("runwise", "global", "none"),
  nuisance_check = c("warn", "error", "drop", "none")
)
```

Arguments

<code>degree</code>	Integer drift degree passed to <code>baseline_model()</code> .
<code>basis</code>	Drift basis: one of "bs", "poly", "ns", "constant".
<code>confounds</code>	Optional confound selection used to populate the per-subject nuisance regressors. Either NULL (no confounds), a character vector of confound column names / patterns, or a <code>bidser</code> confound-set object. Resolved to actual values by <code>from_bids()</code> / <code>instantiate()</code> .
<code>intercept</code>	Intercept handling passed to <code>baseline_model()</code> .
<code>nuisance_check</code>	How to handle problematic nuisance regressors, passed to <code>baseline_model()</code> .

Value

An object of class `baseline_spec`.

See Also

[fmri_template\(\)](#), [baseline_model\(\)](#)

Examples

```
baseline_spec(degree = 3, basis = "bs")
```

batch_errors	<i>Extract per-job errors from a batch result</i>
--------------	---

Description

Extract per-job errors from a batch result

Usage

```
batch_errors(x)
```

Arguments

x An fmri_batch_result.

Value

A named character vector (by job id) of error messages for jobs that failed; empty if all succeeded.

See Also

[run_jobs\(\)](#), [batch_values\(\)](#)

batch_values	<i>Extract successful values from a batch result</i>
--------------	--

Description

Extract successful values from a batch result

Usage

```
batch_values(x)
```

Arguments

x An fmri_batch_result.

Value

A named list (by job id) of reducer outputs for jobs that succeeded.

See Also

[run_jobs\(\)](#), [batch_errors\(\)](#)

build_model	<i>Assemble the full fMRI model for a job</i>
-------------	---

Description

Builds the per-subject baseline_model (from the template's [baseline_spec](#), the dataset's sampling frame, and any nuisance regressors) and combines it with the event model into an fmri_model.

Usage

```
build_model(job, dataset = realize_dataset(job))
```

Arguments

job	An fmri_job .
dataset	The realized dataset (defaults to <code>realize_dataset(job)</code>).

Value

An fmri_model.

See Also

[realize_dataset\(\)](#), [run_job\(\)](#)

coef.fmri_lm	<i>Extract Coefficients from an fmri_lm Fit</i>
--------------	---

Description

Returns estimated coefficients from a fitted fmri_lm model. Every form of the result has one orientation: **voxels x terms**, with one row per voxel (or latent component) and one column per coefficient or contrast, and the term names on the column margin. This matches [stats\(\)](#), [standard_error\(\)](#) and [p_values\(\)](#), so results from the accessor family line up element-wise (for example `coef(fit) / standard_error(fit)`).

Usage

```
## S3 method for class 'fmri_lm'
coef(
  object,
  type = c("betas", "contrasts"),
  include_baseline = FALSE,
  recon = FALSE,
  ...
)
```

Arguments

object	A fitted fmri_lm object.
type	"betas" (default) for regression coefficients, or "contrasts" for the estimates of the simple contrasts defined in the model.
include_baseline	Logical. For type = "betas", also return the baseline (drift, block intercept and nuisance) coefficients, in design matrix column order. The default, FALSE, returns event coefficients only.
recon	Logical; used by latent-space fits to reconstruct voxel-space coefficients. Ignored for ordinary fits.
...	Unused.

Value

For type = "betas", a numeric matrix with one row per voxel and one column per coefficient, columns named after the design matrix (see [coef_names\(\)](#)). For type = "contrasts", a tibble with one row per voxel and one column per contrast.

Orientation change in fmrirege 0.3.0

Before 0.3.0 the default call (type = "betas", include_baseline = FALSE) returned the transpose, terms x voxels, while the other forms were voxels x terms. Code written for the old default that indexed `coef(fit)[term, ]` must now use `coef(fit)[, term]`.

See Also

[coef_names\(\)](#), [stats\(\)](#) (t-statistics), [standard_error\(\)](#), [p_values\(\)](#), [coef_image\(\)](#).

Examples

```
X <- matrix(rnorm(50 * 3), 50, 3)
edata <- data.frame(
  condition = factor(c("A", "B", "A", "B")),
  onsets = c(1, 12, 25, 38),
  run = 1
)
dset <- matrix_frame(X, TR = 2, run_length = 50, event_table = edata)
fit <- fmri_lm(onsets ~ hrf(condition), block = ~run, dataset = dset)
b <- coef(fit) # 3 voxels x 2 event coefficients
dim(b)
colnames(b)
dim(coef(fit, include_baseline = TRUE))
```

coef.fmri_meta	<i>Extract Coefficients from Meta-Analysis</i>
----------------	--

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	<i>Extract Image/Volume for Coefficient</i>
--------------------	---

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 <- matrix_frame(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 NeuroVo1 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 <- matrix_frame(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	<i>Get Available Coefficient Names</i>
------------	--

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 <- matrix_frame(X, TR = 2, run_length = 50,
                     event_table = edata)
fit <- fmri_lm(onsets ~ hrf(condition), block = ~run, dataset = dset)
coef_names(fit)
```

collect_results	<i>Collect per-subject result maps into a group_data object</i>
-----------------	---

Description

Globs the per-subject statistical maps written by [reduce_write_results\(\)](#) under `dir`, pairs them by subject, and builds a [group_data\(\)](#) object for [fmri_meta\(\)](#). When both beta and se maps are present the result carries variance (suitable for inverse-variance meta-analysis); a beta-only result is returned otherwise.

Usage

```
collect_results(
  dir,
  space = NULL,
  beta_desc = "beta",
  se_desc = "se",
  format = c("nifti")
)
```

Arguments

<code>dir</code>	Directory (searched recursively) containing the written maps.
<code>space</code>	Optional BIDS space label to filter on (e.g. "MNI152NLin2009cAsym"); NULL takes all.

beta_desc, se_desc

The BIDS desc- labels of the effect and standard-error maps (defaults "beta" / "se").

format

Group-data backend format (currently "nifti").

Value

A group_data object.

Note

Pairing by BIDS subject requires **fmrigds** with BIDS-aware subject keying. Multi-file "h5" aggregation is not yet supported upstream.

See Also

[reduce_write_results\(\)](#), [run_jobs\(\)](#), [fmri_meta\(\)](#)

Examples

```
## Not run:
tmpl <- fmri_template(onset ~ hrf(trial_type, contrasts = con), ~ run,
                     reducer = reduce_write_results(format = "nifti",
                                                    stats = c("beta", "se"),
                                                    path = "study/glm"))

run_jobs(instantiate(tmpl, manifest))
gd <- collect_results("study/glm", space = "MNI152NLin2009cAsym")
fm <- fmri_meta(gd, ~ 1, method = "fe")

## End(Not run)
```

compute_dvars	<i>Compute DVARs (Temporal Derivative of Timecourses)</i>
---------------	---

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

<code>B</code>	Numeric matrix ($p \times V$) of coefficients.
<code>XtXinv</code>	Numeric matrix ($p \times p$), 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 ($p \times V$), XtX_{inv} ($p \times p$), 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".

compute_spec	<i>Mechanical execution specification</i>
--------------	---

Description

Mechanical execution specification

Usage

```
compute_spec(
  voxel_chunks = 10L,
  backend = c("matrix", "reference"),
  parallel = c("none", "voxels", "chunks"),
  progress = FALSE
)
```

Arguments

voxel_chunks	Number of voxel partitions for joint fitting.
backend	Matrix or formula/reference implementation.
parallel	Parallelization dimension.
progress	Display progress.

contrast	<i>Contrast generic</i>
----------	-------------------------

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	<i>Apply Contrast to Meta-Analysis Results</i>
--------------------	--

Description

The contrast standard error is $\sqrt{c'Var(\hat{\beta})c}$, using the coefficient covariance that `fmri_meta()` stores with every fit. For objects without a stored covariance only single-coefficient contrasts are exact; other contrasts fall back to a diagonal approximation with a warning.

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., <code>~ groupold - groupyoung</code>) • A named vector (e.g., <code>c("groupold" = 1, "groupyoung" = -1)</code>)
...	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	<i>Create a contrast set</i>
--------------	------------------------------

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.fmri_model	<i>correlation_map.fmri_model</i>
----------------------------	-----------------------------------

Description

Heatmap of pairwise correlations between the columns of an `fmri_model`'s combined event + base-line design matrix, grouped into panels by model term (event terms, drift, nuisance).

By default (`within_run = TRUE`) each column is centred within each run and the run-intercept columns are dropped before correlating. This shows the collinearity that matters for estimation once run means are modelled, and removes spurious correlations that zero-padding creates between columns belonging to different runs. Cells for pairs of columns that live in different runs are left blank.

Usage

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

Arguments

<code>x</code>	An <code>fmri_model</code> .
<code>method</code>	Correlation method (e.g., "pearson", "spearman").
<code>half_matrix</code>	Logical; if TRUE (default), display only the lower triangle.
<code>absolute_limits</code>	Logical; if TRUE, the colour scale spans -1 to 1; otherwise it spans plus or minus the largest absolute correlation shown.
<code>within_run</code>	Logical; centre columns within runs and drop run intercepts before correlating (default TRUE). FALSE correlates the raw columns.
<code>label_values</code>	Logical or NULL; print correlations in the cells. NULL (default) labels every cell when there are at most 12 columns, and otherwise only cells with $ r \geq 0.3$ that involve an event regressor.
<code>...</code>	Additional arguments passed to geom_tile .

Value

A `ggplot2` object.

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)
```

dataset_spec	<i>Describe how to (re)construct a subject's dataset</i>
--------------	--

Description

A small, serializable recipe for a dataset: the name of a dataset constructor plus the arguments to call it with. For file-backed data the arguments are paths and run lengths (no voxel data), which keeps the enclosing [fmri_job](#) tiny and portable. The dataset is realized lazily on the worker.

Usage

```
dataset_spec(constructor, args = list(), source = c("file", "inline"))
```

Arguments

constructor Name of a frame constructor (a string): one of "nifti_frame", "matrix_frame", "neurovec_frame", or "latent_frame". Resolved at run time, so the data is not loaded when the spec is built.

args A named list of arguments passed to constructor (for "nifti_frame": scans, TR, run_length, event_table, mask, base_path, ...).

source Either "file" (paths; nothing loaded until run) or "inline" (data already in args, e.g. a matrix for matrix_frame).

Value

An object of class dataset_spec.

See Also

`fmri_job()`, `nifti_frame()`, `matrix_frame()`

Examples

```
dataset_spec("nifti_frame",
  args = list(scans = c("run-1_bold.nii.gz", "run-2_bold.nii.gz"),
    TR = 2, run_length = c(200, 200)),
  source = "file")
```

`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 of an `fmri_model`, which merges the `event_model` and `baseline_model` regressors. Rows are scans (time runs downward); columns are regressors, kept in model order and grouped into panels by term (event terms, drift, intercept, nuisance). Run boundaries are marked with horizontal rules, and the subtitle reports the rank and condition number of the design.

By default each column is divided by its maximum absolute value, so regressors on very different scales (an intercept, a motion trace, a convolved event train) are all legible on one colour scale. Set `scale_columns = FALSE` to show raw values. Run-intercept columns are drawn in grey, since they carry no information beyond run membership.

The condition number is computed after scaling every column to unit length, so it reflects collinearity rather than the units of the columns.

Usage

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

Arguments

`x` An `fmri_model` object.

`block_separators` Logical; if TRUE, draw rules between runs.

`rotate_x_text` Logical; if TRUE, rotate x-axis labels to vertical.

fill_midpoint	Numeric or NULL; centre of the diverging colour scale. Defaults to 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.
scale_columns	Logical; if TRUE (default), scale each column to a maximum absolute value of 1.
...	Additional arguments passed to geom_raster .

Value

A ggplot2 plot object.

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

<code>x</code>	An <code>fmri_model</code> object
<code>blockid</code>	Optional numeric vector specifying which blocks/runs to include
<code>...</code>	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

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

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	<i>List Registered Engine Specifications</i>
--------------	--

Description

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

Usage

```
engine_specs()
```

Value

A named list of `fmrireg_engine_spec` objects.

estimate	<i>Deprecated helper: estimate()</i>
----------	--------------------------------------

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	<i>Estimate Beta Coefficients for fMRI Data</i>
----------------	---

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 using various regression methods including mixed models, least squares, and PLS.

Usage

```
estimate_betas(x, ...)

## S3 method for class 'fmri_frame'
estimate_betas(
  x,
  fixed = NULL,
  ran,
  block,
  method = c("mixed", "lss", "ols"),
  basemod = NULL,
  maxit = 1000,
  fracs = 0.5,
  progress = TRUE,
  ...
)
```

Arguments

x	An <code>fmri_frame</code> (see <code>matrix_frame()</code> , <code>neurovec_frame()</code> , <code>nifti_frame()</code> , and <code>latent_frame()</code>). When the frame's feature space is a <code>volume_space</code> , the fixed and random betas are returned as <code>NeuroVec</code> objects on that grid; otherwise (index or basis spaces) they are returned as coefficient-by-feature matrices.
...	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.

<code>method</code>	The regression method for estimating trialwise betas; one of "mixed", "lss", or "ols".
<code>basemod</code>	A <code>baseline_model</code> instance to regress out of data before beta estimation (default: NULL).
<code>maxit</code>	Maximum number of iterations for optimization methods (default: 1000).
<code>fracs</code>	Fraction of voxels used for prewhitening.
<code>progress</code>	Logical; show progress bar.

Details

This is a generic function whose `fmri_frame` method adapts to the frame's feature space: volumetric frames (`volume_space`) return `NeuroVec` betas, while matrix-format (`index_space`) and latent (`basis_space`) frames return coefficient matrices.

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`: fixed effect betas (`NeuroVec` for volumetric frames, matrix otherwise).
- `betas_ran`: random effect betas (`NeuroVec` for volumetric frames, matrix otherwise).
- `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).

References

- Mumford, J. A., et al. (2012). Deconvolving BOLD activation in event-related designs for multi-voxel 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

[matrix_frame](#), [neurovec_frame](#), [latent_frame](#)
[matrix_frame](#), [baseline_model](#), [event_model](#)

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 <- matrix_frame(
  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"
)

## 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 <- nifti_frame(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_hrf
```

```
Estimate smooth condition-level hemodynamic response functions
```

Description

`estimate_hrf()` estimates condition-level HRF curves directly from an fMRI dataset. It builds an event-aligned spline basis, removes baseline and fixed nuisance effects once, and solves every voxel together with one penalized multiresponse linear system. This replaces the former voxel-by-voxel GAM implementation, whose predictor did not represent post-stimulus time.

Usage

```
estimate_hrf(
  form,
  fixed = NULL,
  block,
  dataset,
  bs = NULL,
  rsam = seq(0, 20, by = 1),
  basemod = NULL,
  k = 8L,
  fx = NULL,
  progress = FALSE,
  basis = c("bspline", "tent"),
  lambda = "gcv",
  lambda_grid = c(0, 10^seq(-4, 4, length.out = 25L)),
  ci_level = 0.95
)
```

Arguments

<code>form</code>	A two-sided event-model formula containing one or more <code>hrf()</code> terms. <code>trialwise()</code> terms are rejected because this function estimates condition-level curves.
<code>fixed</code>	Optional event-model formula whose design is treated as nuisance.
<code>block</code>	Formula identifying acquisition runs or blocks.
<code>dataset</code>	An <code>fmri_frame</code> .
<code>bs</code>	Deprecated legacy basis selector. Values from the former GAM API are mapped to <code>basis = "bspline"</code> .
<code>rsam</code>	Strictly increasing, finite post-stimulus times beginning at zero.
<code>basemod</code>	Optional baseline model. The default is a constant baseline.
<code>k</code>	Number of free HRF basis functions per curve. At least four for the default cubic B-spline basis.

<code>fx</code>	Deprecated legacy selector. <code>fx = TRUE</code> maps to <code>lambda = 0</code> when <code>lambda</code> is omitted.
<code>progress</code>	Show progress while scanning the GCV grid.
<code>basis</code>	Either "bspline" (cubic) or "tent" (piecewise linear).
<code>lambda</code>	Non-negative smoothing strength or "gcv" for shared automatic selection.
<code>lambda_grid</code>	Candidate smoothing strengths used for GCV.
<code>ci_level</code>	Confidence level in (0, 1), or NULL to omit intervals.

Details

A single smoothing parameter is shared across voxels. With `lambda = "gcv"`, it is selected by scale-normalized generalized cross-validation so high- variance voxels do not dominate the choice. Curves are constrained to zero at the beginning and end of `rsam`. Standard errors and confidence intervals use the fitted penalized-linear-model covariance under independent, homoscedastic time-point errors; they are not autocorrelation-robust.

Value

An object of class `fmri_hrf_estimate`. Its `estimate` and `std.error` arrays have dimensions time by curve by voxel. The object also contains labeled curve metadata, basis coefficients, smoothing diagnostics, and the designs used for estimation. Use `tidy()`, `predict()`, `coef()`, or `as.matrix()` for common downstream representations.

See Also

[fitted_hrf\(\)](#), [event_model\(\)](#), [baseline_model\(\)](#)

Examples

```
set.seed(18)
n <- 80L
events <- data.frame(
  onset = seq(6, 62, by = 8),
  condition = factor(rep(c("A", "B"), 4)),
  run = 1L
)
dataset <- matrix_frame(
  matrix(rnorm(n * 2), nrow = n),
  TR = 1,
  run_length = n,
  event_table = events
)
fit <- estimate_hrf(
  onset ~ hrf(condition),
  block = ~run,
  dataset = dataset,
  rsam = 0:12,
  k = 6,
  lambda = 1
)
```

```
tidy(fit, voxel = 1)
```

estimation_spec	<i>Statistical fitting scope</i>
-----------------	----------------------------------

Description

Statistical fitting scope

Usage

```
estimation_spec(
  scope = c("joint", "runwise_meta"),
  run_pooling = "fixed_effect"
)
```

Arguments

scope	Fit a single joint model or fit runs separately and combine them by fixed-effect meta-analysis.
run_pooling	Runwise combination rule. Only fixed-effect pooling is currently supported.

evaluate_method_performance	<i>Evaluate Method Performance on Benchmark Dataset</i>
-----------------------------	---

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)
```

export_jobs

Export jobs for external / array execution

Description

Writes a serialized job manifest plus a minimal, backend-agnostic `run_one.R` runner to `dir`. Deliberately emits no scheduler code: drive it with whatever array system you have, passing the 1-based job index as the first argument, e.g. `Rscript run_one.R $SLURM_ARRAY_TASK_ID`.

Usage

```
export_jobs(
  jobs,
  dir,
  overwrite = FALSE,
  setup = "library(fmrireg)",
  results_dir = "results"
)
```

Arguments

<code>jobs</code>	A single fmri_job or a list of them (e.g. from instantiate()).
<code>dir</code>	Output directory (created if needed).
<code>overwrite</code>	Overwrite an existing manifest. Default <code>FALSE</code> .
<code>setup</code>	Character vector of R lines run at the top of <code>run_one.R</code> before jobs are executed (load packages, set threads, etc.). Default loads fmrireg .
<code>results_dir</code>	Default output subdirectory written by <code>run_one.R</code> .

Value

Invisibly, a list describing what was written (dir, manifest, runner, n, ids).

See Also

`read_jobs()`, `run_jobs()`, `instantiate()`

Examples

```
## Not run:
jobs <- instantiate(tmpl, manifest)
export_jobs(jobs, "study/jobs")
# then, per array task: Rscript study/jobs/run_one.R $SLURM_ARRAY_TASK_ID

## End(Not run)
```

extract_csv_data	<i>Extract Data for Meta-Analysis from CSV</i>
------------------	--

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	<i>Extract Nuisance Timeseries from Mask</i>
-----------------------------	--

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_frame 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	<i>Fit Contrasts</i>
---------------	----------------------

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, cov.unscaled = NULL, ...)
```

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).
cov.unscaled	Optional unscaled coefficient covariance $(X'X)^{-1}$ of object, to reuse across several contrasts on the same fit (default: computed from object).
...	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)

Details

Multi-run fits estimated run by run (runwise_meta) with temporal noise models, robust weighting, volume weights or projection whiten or weight each run differently, so their pooled betas have no single stored covariance; post-hoc contrasts are refused for them. Specify such contrasts in the model formula, or fit with estimation_spec(scope = "joint").

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 <code>fmri_model</code> describing the design.
XtX	$p \times p$ cross-product of the design matrix.
XtS	$p \times V$ cross-product of design with data.
StS	length- V vector of sum of squares per voxel.
df	Residual degrees of freedom.
cfg	Optional <code>fmri_lm_control</code> ; used for metadata only.
dataset	Optional dataset whose feature axis describes the columns of <code>XtS</code> . It is never inferred from <code>model\$dataset</code> .
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

This helper performs IID, non-robust OLS inference. It rejects controls that request AR or robust fitting; use `fit_glm_with_config()` for those models.

Usage

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

Arguments

model	An <code>fmri_model</code> describing the design.
Y	Numeric matrix with <code>nrow(Y)</code> time points and columns matching voxels.
cfg	Optional <code>fmri_lm_control</code> ; defaults to <code>fmri_lm_control()</code> and must request IID noise and no robust fitting.
dataset	Optional dataset whose feature axis describes the columns of <code>Y</code> . It is never inferred from <code>model\$dataset</code> , because a transformed response can have a different feature space.
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`.

<code>fit_glm_with_config</code>	<i>Fit GLM with full config (AR/robust) on a transformed series</i>
----------------------------------	---

Description

Runs the same integrated solver used by `fmri_lm`, honoring AR/robust options from `cfg`, but on an externally provided response matrix `Y` ($T \times V$). 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

<code>model</code>	An <code>fmri_model</code> describing the design.
<code>Y</code>	Numeric matrix with <code>nrow(Y)</code> time points and columns matching voxels.
<code>cfg</code>	Optional <code>fmri_lm_control</code> ; defaults to <code>fmri_lm_control()</code> .
<code>dataset</code>	Optional dataset whose feature axis describes the columns of <code>Y</code> . It is never inferred from <code>model\$dataset</code> , because a transformed response can have a different feature space.
<code>strategy</code>	Character label recorded on the returned object. Defaults to "external".
<code>engine</code>	Character label indicating the engine that produced the fit. Defaults to "external".

Value

An object of class `fmri_lm`.

<code>fitted_hrf</code>	<i>fitted_hrf</i>
-------------------------	-------------------

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

<code>x</code>	An object for which the fitted HRF should be computed
<code>sample_at</code>	A vector of time points at which the HRF should be sampled
<code>...</code>	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 <- matrix_frame(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	<i>Compute Fitted Hemodynamic Response Functions for an fmri_lm Object</i>
--------------------	--

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	<i>Flip Sign of Coefficients</i>
-----------	----------------------------------

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	<i>Benchmark fMRI datasets</i>
-------------------------	--------------------------------

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_hrf_estimate-methods

Methods for smooth HRF estimates

Description

Methods for smooth HRF estimates

Usage

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

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

## S3 method for class 'fmri_hrf_estimate'
as.matrix(x, curve = NULL, what = c("estimate", "std.error"), ...)

## S3 method for class 'fmri_hrf_estimate'
predict(object, newdata = object$time, se.fit = FALSE, ...)
```

Arguments

x, object	An fmri_hrf_estimate object.
...	Unused.
curve	A curve name or one-based curve index. Required by as.matrix() when the object contains multiple curves.
what	Return estimated curves or their standard errors.
newdata	Numeric post-stimulus times within the fitted span.
se.fit	Return prediction standard errors with fitted curves.

fmri_job

*Construct a per-subject job recipe***Description**

Binds an [fmri_template](#) to one subject's [dataset_spec](#). The result is a fully serializable recipe (no voxel data, no captured environment) that [run_job\(\)](#) turns into a fitted model and reduced output. Build jobs with [instantiate\(\)](#) rather than by hand in the common case.

Usage

```
fmri_job(id, template, dataset_spec, meta = list(), nuisance = NULL)
```

Arguments

id	A unique job identifier (e.g. a subject label).
template	An fmri_template .
dataset_spec	A dataset_spec describing this subject's data.
meta	Optional named list of metadata used for output keying (e.g. <code>list(subject = "01", task = "stroop", space = "MNI152...")</code>).
nuisance	Optional per-subject nuisance / confound regressors fed to the baseline model at run time: <code>NULL</code> , a numeric matrix with one row per scan (split across runs), or a list of per-run matrices. (A deferred file-read spec is also accepted by run_job() .)

Value

An object of class `fmri_job`.

See Also

[instantiate\(\)](#), [run_job\(\)](#), [export_jobs\(\)](#)

Examples

```
tmpl <- fmri_template(onset ~ hrf(condition), ~ run)
ds <- dataset_spec("nifti_frame",
  args = list(scans = "run-1_bold.nii.gz", TR = 2,
    run_length = 200), source = "file")
fmri_job("sub-01", tmpl, ds, meta = list(subject = "01"))
```

fmri_latent_lm	<i>Fast fMRI Regression Model Estimation from a Latent Component Dataset</i>
----------------	--

Description

This function estimates a regression model for fMRI data using a latent component dataset. The dataset must be an `fmri_frame` whose feature space is a `fmridataset::basis_space` (see [latent_frame\(\)](#)), so that the assay holds component scores and the space carries the spatial loadings.

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

<code>formula</code>	A formula specifying the regression model.
<code>block</code>	A factor indicating the block structure of the data.
<code>baseline_model</code>	An optional baseline model.
<code>dataset</code>	A latent <code>fmri_frame</code> (see latent_frame()).
<code>durations</code>	The duration of events in the dataset.
<code>drop_empty</code>	Whether to drop empty events from the model. Default is TRUE.
<code>robust</code>	Must be FALSE. Robust weights are voxel-specific and cannot be computed on component scores, so any robust request is an error.
<code>autocor</code>	The autocorrelation model. 'none' (OLS), 'ar1' or 'ar2'. AR models use one set of AR coefficients per run shared across voxels, estimated so that the fit equals a dense fit of the reconstructed voxel data. 'auto' and 'arma' are not implemented and are an error.
<code>bootstrap</code>	Must be FALSE; bootstrapping is not implemented for latent datasets and TRUE is an error.
<code>nboot</code>	Unused (see <code>bootstrap</code>).
<code>...</code>	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.

The public fitting boundary separates statistical choices in `fmri_lm_control()` from mechanical execution choices in `compute_spec()`. Optional engines use the same control object and receive their private arguments through `engine_args`.

Usage

```
fmri_lm(formula, ...)

## S3 method for class 'formula'
fmri_lm(
  formula,
  block,
  baseline_model = NULL,
  dataset,
  durations = 0,
  drop_empty = TRUE,
  control = fmri_lm_control(estimation = estimation_spec("runwise_meta")),
  compute = compute_spec(),
  engine = NULL,
  engine_args = list(),
  lowrank = NULL,
  ...)
```

```

)

## S3 method for class 'fmri_model'
fmri_lm(
  formula,
  dataset = NULL,
  control = fmri_lm_control(estimation = estimation_spec("runwise_meta")),
  compute = compute_spec(),
  engine = NULL,
  engine_args = list(),
  lowrank = NULL,
  ...
)

```

Arguments

formula	A model formula describing the event structure or an <code>fmri_model</code> object.
...	Transitional legacy arguments retained for one compatibility window. New code must express statistical choices in <code>control</code> and execution choices in <code>compute</code> ; unknown arguments are rejected.
block	Formula describing run/block structure.
baseline_model	Optional baseline/nuisance model.
dataset	An <code>fmri_frame</code> . For an <code>fmri_model</code> method this may be omitted when the model already owns its dataset.
durations	Event durations passed to model construction.
drop_empty	Remove empty factor levels during model construction.
control	A validated <code>fmri_lm_control()</code> .
compute	A validated <code>compute_spec()</code> .
engine	Optional registered engine name.
engine_args	Named list passed only to engine.
lowrank	Optional <code>lowrank_control()</code> for the latent-sketch engine.

Value

An object of class `fmri_lm`.

fmri_lm_control	<i>Configure an fMRI linear model</i>
-----------------	---------------------------------------

Description

`fmri_lm_control()` is the single statistical configuration boundary for `fmri_lm()`. Each section is a validated typed specification; mechanical choices such as chunking and parallelism belong in `compute_spec()`.

Usage

```
fmri_lm_control(
  estimation = NULL,
  noise = NULL,
  robust = NULL,
  variance = NULL,
  weights = NULL,
  projection = NULL,
  na_action = c("error", "propagate"),
  ...
)
```

Arguments

estimation	An <code>estimation_spec()</code> .
noise	A <code>noise_spec()</code> .
robust	A <code>robust_spec()</code> .
variance	A <code>variance_spec()</code> .
weights	A <code>weights_spec()</code> .
projection	A <code>projection_spec()</code> .
na_action	Either "error" or "propagate".
...	Transitional legacy option lists retained for one compatibility window. New code should use the typed sections above.

Value

An object of class `fmri_lm_control`.

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": Not implemented. Accepted for backward compatibility; it runs the Paule-Mandel estimator, says so with a message, and the fit records method = "pm".
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). The subject-level variances are taken as known and method chooses the between-subject variance estimator. • "equal": Equal weights for all subjects • "custom": User-provided weights (must supply weights argument) "equal" and "custom" weights are <i>relative</i>: the model is weighted least squares, $Var(y_i) = \sigma^2/w_i$, with the residual scale σ^2 estimated per feature (as <code>lm(weights = w)</code> does; equal weights give the one-sample / OLS group t-statistic). The subject-level standard errors and method are then not used, and tau2, Q and I2 are NA.
weights_custom	Numeric vector or matrix of positive 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 (analysed voxels or ROIs).
combine	For t-statistic-only data, combination method ("stouffer", "fisher", or "lancaster"). Stouffer combines z-scores and supports equal or custom weighting (via weights); inverse-variance weights are unavailable without effect standard errors. 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; kept for backward compatibility. The packed upper-triangular Var(beta) per feature is now always returned under \$cov (for subject-level effect data) so that contrast() gives exact standard errors.
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, groupB = 1))

# Exact fit-time contrast without storing covariance
fit_con <- fmri_meta(gd, formula = ~ 1 + group, method = "pm",
                    contrasts = c("(Intercept)" = 0, groupB = 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("fmrirege.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). "reml" is not implemented: it is accepted for backward compatibility, runs Paule-Mandel with a message, and the result records method = "pm".
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 $c'(X'WX)^{-1}c$ 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 <code>fmri_meta_fit</code>
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). "reml" is not implemented: it is accepted for backward compatibility, runs Paule-Mandel with a message, and the result records <code>method = "pm"</code> .
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`

<code>fmri_meta_fit_cov</code>	<i>Fit Meta-Analysis and return packed covariance per voxel</i>
--------------------------------	---

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("fmriregh.num_threads", 0)
)
```

Arguments

<code>Y</code>	Numeric matrix of effect sizes (subjects x features)
<code>V</code>	Numeric matrix of variances (subjects x features)
<code>X</code>	Numeric matrix; design matrix (subjects x predictors), including intercept
<code>method</code>	Character scalar; meta-analysis method: "pm" (Paule-Mandel), "dl" (DerSimonian-Laird), "fe" (fixed-effects). "reml" is not implemented: it is accepted for backward compatibility, runs Paule-Mandel with a message, and the result records <code>method = "pm"</code> .
<code>robust</code>	Character scalar; robust estimation method: "none" or "huber"
<code>huber_c</code>	Numeric scalar; tuning constant for Huber M-estimator (default: 1.345). Smaller values provide more robust estimates but may reduce efficiency.
<code>robust_iter</code>	Integer scalar; number of IRLS iterations for robust estimation (default: 2)
<code>n_threads</code>	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	<i>Construct an fMRI Regression Model</i>
------------	---

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_frame 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	<i>OLS Fit with Optional Voxelwise Covariate</i>
--------------	--

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	Formula describing run/block structure.
baseline_model	Optional baseline/nuisance model.
dataset	An fmri_frame. For an fmri_model method this may be omitted when the model already owns its dataset.
durations	Event durations passed to model construction.
drop_empty	Remove empty factor levels during model construction.
strategy	Fit runs separately and combine them, or fit a joint model partitioned into voxel chunks.
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 <- matrix_frame(mat, TR=1, run_length=c(50,50),event_table=etab)
dset2 <- matrix_frame(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_template

Define a subject-invariant model template

Description

Captures a complete model specification that is constant across subjects. Per-subject data (BOLD paths, event table, run lengths, confounds) is bound later with [instantiate\(\)](#) to produce serializable [fmri_job](#) recipes that can be executed locally, in parallel via **future**, or on a cluster.

Usage

```
fmri_template(
  formula,
  block,
  baseline = baseline_spec(),
  durations = 0,
  contrasts = NULL,
  control = fmri_lm_control(estimation = estimation_spec("runwise_meta")),
  compute = compute_spec(voxel_chunks = 1L),
  engine = NULL,
  engine_args = list(),
  reducer = NULL
)
```

Arguments

formula	Event model formula, e.g. <code>onset ~ hrf(condition)</code> .
block	Block / run-structure formula, e.g. <code>~ run</code> .
baseline	A baseline_spec() describing the nuisance model.
durations	Event durations passed through to the event model.
contrasts	Optional contrast specification (e.g. from <code>contrast_set()</code>).
control	An <code>fmri_lm_control</code> from fmri_lm_control() holding the robust / AR / pre-processing options for fitting.
compute	A compute_spec() describing backend, voxel partitioning, parallelization, and progress reporting. Statistical runwise versus joint fitting belongs in <code>control\$estimation</code> .
engine	Optional fitting engine name (see register_engine()).
engine_args	Optional list of engine arguments.
reducer	Optional function <code>function(fit, job)</code> that turns a fitted <code>fmri_lm</code> into the per-subject output (written to disk and/or returned). <code>NULL</code> (the default) returns the fitted object unchanged. See builtins such as reduce_write_results() .

Details

The template is pure, serializable data: it holds no voxel data and (by design) no captured execution environment. The reducer is the one field that can accidentally capture state; see **Reducer serializability** below.

Value

An object of class `fmri_template`.

Reducer serializability

A reducer should be a top-level / package function (or a closure that captures nothing), so it survives serialization to a worker node. A closure that captures local variables will drag those bindings along when the job is written with `saveRDS()`; `fmri_template()` warns in that case.

See Also

`baseline_spec()`, `instantiate()`, `fmri_lm_control()`

Examples

```
tmpl <- fmri_template(onset ~ hrf(condition), ~ run,
                     baseline = baseline_spec(degree = 3))
tmpl
```

fmri_ttest

fmrirege 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 `fmrirege` 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

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

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"
))
```

from_bids	<i>Build a manifest from a BIDS project</i>
-----------	---

Description

Uses **bidser** to discover preprocessed BOLD scans, events, confounds, TR, and run lengths for each subject of a BIDS dataset, returning an `fmri_manifest` (a list of per-subject bindings) suitable for [instantiate\(\)](#).

Usage

```
from_bids(
  proj,
  task,
  space = NULL,
  confounds = NULL,
  mask = NULL,
  desc = "preproc",
  subjects = NULL,
  ...
)
```

Arguments

proj	A <code>bidser::bids_project</code> (opened with <code>fmriprep = TRUE</code> for derivative discovery).
task	BIDS task label.
space	BIDS space for preprocessed scans (e.g. "MNI152NLin2009cAsym"); NULL matches any.

confounds	Confound selection passed to <code>bidser::read_confounds()</code> as <code>cvars</code> (character vector or a <code>confound_set</code>); NULL reads no confounds.
mask	A brain-mask path (length-1 character) applied to every subject's file-backed dataset. Required for file-backed BIDS data, which needs a mask (auto-resolution of the fmripred brain mask is a planned enhancement).
desc	BIDS desc- label of the preprocessed scans (default "preproc").
subjects	Optional character vector of subject labels (bare ids, e.g. "01"); default is all participants.
...	Forwarded to <code>bidser::read_confounds()</code> (e.g. <code>npcs</code> , <code>clean</code> , <code>na_action</code>).

Details

The design formula's variables must match the columns of the BIDS `events.tsv` (e.g. `trial_type`); a `run` column is added for the block structure. Confound columns are selected by `confounds` (a character vector or `bidser::confound_set()` result) passed straight to `bidser::read_confounds()`.

Value

An `fmri_manifest`: a list of bindings, one per subject.

See Also

[as_manifest\(\)](#), [instantiate\(\)](#), [preflight\(\)](#)

Examples

```
## Not run:
proj <- bidser::bids_project("study/", fmripred = TRUE)
mani <- from_bids(proj, task = "stroop", space = "MNI152NLin2009cAsym",
  confounds = bidser::confound_set("motion6"),
  mask = "study/derivatives/.../space-MNI..._desc-brain_mask.nii.gz")
jobs <- instantiate(template, mani)

## End(Not run)
```

generate_interaction_contrast

Fast factorial contrast generators

Description

Returns a matrix **N_cells x N_contrasts** - row *r* is the design cell in row *r* of `des` (for `expand.grid()` input, the first factor varies fastest, which is also the column order of `fmridesign` crossed terms such as `hrf(A, B)`), 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	<i>Get Available Contrasts</i>
---------------	--------------------------------

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	<i>Get Covariates</i>
----------------	-----------------------

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	<i>Get Available ROIs</i>
----------	---------------------------

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	<i>Get Subject IDs</i>
--------------	------------------------

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

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

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 an fmri_frame with event table
dset <- matrix_frame(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

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

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_1ss()` 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 an fmri_frame with event table
dset <- matrix_frame(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 "fmriilm": List of fmri_ilm objects
format	Character string specifying the input format. One of "auto" (default), "h5", "nifti", "csv", or "fmriilm". 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_ilm
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

<code>data</code>	Either a path to a CSV file or a data frame containing the data
<code>effect_cols</code>	Named vector or list specifying column names for effect statistics. E.g., <code>c(beta = "mean_activation", se = "std_error")</code> or <code>c(t = "t_stat", df = "df")</code>
<code>subject_col</code>	Character string specifying the column containing subject IDs
<code>roi_col</code>	Character string specifying the column containing ROI names (optional)
<code>contrast_col</code>	Character string specifying the column containing contrast names (optional)
<code>covariate_cols</code>	Character vector of column names to use as covariates (optional)
<code>wide_format</code>	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_fmriilm

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_fmriilm(
  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	"gram" (inner products of the design rows) or "cosine" (the same, scaled to unit diagonal). With normalise = TRUE both give $K[i, j] / \sqrt{K[i, i] * K[j, j]}$, a symmetric matrix with unit diagonal.

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, overwrite = FALSE, commands = NULL)
```

Arguments

dest_dir	Destination directory for installed command wrappers. Required: there is no default, so nothing is written to the home directory unless asked (e.g. <code>install_cli("~/local/bin")</code>).
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)
```

instantiate

Instantiate a template into per-subject jobs

Description

Binds an [fmri_template](#) to one or more per-subject data bindings, producing serializable [fmri_job](#) recipes. No data is loaded: file-backed scans are kept as paths and realized lazily by [run_job\(\)](#).

Usage

```
instantiate(template, x, ...)
```

Arguments

template	An fmri_template .
x	A single binding (a named list with at least <code>id</code>), a list of bindings, or a manifest <code>data.frame</code> (one row per analysis unit, with list-columns for scans/events/run_length/etc.).
...	Unused.

Value

A single [fmri_job](#) if `x` is one binding, otherwise a list of [fmri_jobs](#).

See Also

[fmri_job\(\)](#), [run_job\(\)](#), [from_bids\(\)](#)

Examples

```
tmpl <- fmri_template(onset ~ hrf(condition), ~ run)
b <- list(id = "sub-01", scans = matrix(rnorm(80 * 3), 80, 3),
          TR = 2, run_length = c(40, 40),
          events = data.frame(onset = c(5, 25, 45, 65),
                              condition = factor(c("A", "B", "A", "B")),
                              run = c(1, 1, 2, 2)))
job <- instantiate(tmpl, b)
```

latent_frame	<i>Build an fmri_frame from a latent (basis) decomposition</i>
--------------	--

Description

A `fmristore::LatentNeuroVec` stores temporal scores (time by component) and spatial loadings (voxel by component). This helper places the scores in the assay and the loadings in a `fmridataset::basis_space()` whose parent is the mask's `volume_space`, so `fmrireg`'s latent engines can fit in component space and reconstruct voxel maps through `fmridataset::basis_synthesis()`.

Usage

```
latent_frame(
  x,
  TR,
  run_length = NULL,
  event_table = NULL,
  censor = NULL,
  assay = "scores"
)
```

Arguments

<code>x</code>	A <code>fmristore::LatentNeuroVec</code> , or a list with elements <code>basis</code> (time by component scores), <code>loadings</code> (masked-voxel by component), and <code>mask</code> (a <code>neuroim2::LogicalNeuroVol</code>).
<code>TR</code>	Repetition time in seconds; one value, or one per run.
<code>run_length</code>	Volumes per run; defaults to a single run spanning all score rows.
<code>event_table, censor</code>	As in <code>matrix_frame()</code> .
<code>assay</code>	Name of the assay.

Value

An `fmri_frame` over a synthesis-only `basis_space`.

See Also

`matrix_frame()`, `fmridataset::basis_space_from_decoder()`

`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 fmrirege 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: Reconstructed true HRF object(s) plus complete, versioned generation recipes
- oracle_contract: Scope and tolerance of the valid stored numerical oracle, including limitations for partial-truth datasets
- 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
- run_length: Number of acquired time points
- 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
```

lowrank_control	<i>Low-rank / sketch controls for fast GLM</i>
-----------------	--

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),
  ncomp = NULL,
  noise_pcs = 0L
)
```

Arguments

parcels	Optional parceling, e.g., a <code>neuroim2::ClusteredNeuroVol</code> 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 controlling the temporal sketch, with elements

method "gaussian" (default), "countsketch" or "srht" (subsampled randomized Hadamard transform). "ihs" (iterative Hessian sketch) is deprecated: it now computes the exact OLS fit and emits a once-per-session message.

m Number of sketch rows; NULL uses $\min(8 * p, T)$ for a design with p columns and T scans. The sketch-and-solve methods require $m > p$; "ihs" ignores m.

The former "ihs" controls iters and tol are accepted and ignored.

All sketches are normalised ($E\|Sr\|^2 = \|r\|^2$), so residual variances, rss and covariances are reported on the data scale. "gaussian", "countsketch" and "srht" are sketch-and-solve estimators: they fit the model to m sketched rows, and their standard errors are the conditional-on-sketch standard errors of that estimator (roughly $\sqrt{T / m}$ times the full-data OLS ones), with Satterthwaite residual degrees of freedom (about m - p). "ihs" returns exact OLS coefficients, covariance, residual variance and T - rank degrees of freedom. With many response columns and few design columns, one exact least-squares pass costs a single $X'Z$ product, which no iterative sketch can undercut; "ihs" was never faster than exact OLS here.

For speed, prefer "countsketch" or "gaussian". With p = 20, m = 8p and r = 5e4 response columns, a sketch-and-solve fit took 0.04-0.06 s for T from

400 to 1000 scans, against 0.05-0.14 s for exact OLS (Apple M3 Max, Accelerate BLAS). "srht" took 0.07-0.09 s: its fast Walsh-Hadamard transform costs $O(T \log T)$ per response column, CountSketch touches each scan once ($O(T)$), and the Gaussian sketch's $O(mT)$ dense product runs at BLAS speed, so "srht" loses to both at these sizes and overtakes exact OLS only for longer series. Its advantage is an orthonormal-row sketch ($SS' = (T/m)I$ when T is a power of two). The SRHT kernel runs in parallel over response columns with `getOption("fmrireg.num_threads")` threads when the package is built with OpenMP.

For sketch-and-solve fits, `result$rss` is the residual sum of squares of the `m` sketched rows. Its expectation is $\sigma^2 \kappa$, with $\kappa = \text{tr}(PSS')$ stored in `result$sketch$kappa`, so the residual variance is `rss / kappa`, not `rss / rdf`. For "ihs", `kappa` equals the residual degrees of freedom. With landmarks, `rss` is interpolated from the landmark voxels, like the residual variance.

Estimability is judged on the full (whitened) design with the same pivoted-QR rule that exact fits use. Aliased coefficients are reported as NA, and contrasts that load on them are NA with a warning. If the sketched design restricted to the estimable columns is numerically rank deficient, `m` is too small for the design and the fit stops with an error rather than regularising.

`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".

<code>ma_parameters</code>	<i>Extract Estimated MA Parameters from fmri_lm Fit</i>
----------------------------	---

Description

Retrieves the estimated moving-average parameters from a fitted fMRI linear model that used ARMA or MA temporal-noise modeling.

Usage

```
ma_parameters(object, ...)  
  
## S3 method for class 'fmri_lm'  
ma_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 coefficients averaged across runs, "per_run" returns the run-level estimates, and "raw" returns the stored structure without post-processing.

Value

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

matrix_frame	<i>Build an fmri_frame from an in-memory matrix</i>
--------------	---

Description

fmrireg fits models to `fmridataset::fmri_frame()` objects. This helper builds one from a time-by-feature matrix plus the run structure and event table that the design machinery needs. Observation IDs are derived deterministically as `run-<r>-vol-<index>`; feature IDs default to the matrix column names when they are unique and to `feature-<j>` otherwise.

Usage

```
matrix_frame(
  datamat,
  TR,
  run_length,
  event_table = NULL,
  censor = NULL,
  feature_ids = NULL,
  assay = "signal"
)
```

Arguments

datamat	A numeric matrix with one row per acquired volume and one column per feature (voxel, vertex, parcel, or component).
TR	Repetition time in seconds; one value, or one per run.
run_length	Integer vector giving the number of volumes in each run. Must sum to <code>nrow(datamat)</code> .
event_table	Optional data frame of events (onsets, conditions, and the block variable). It is stored on the frame as a keyed <code>fmridataset::event_table()</code> ; an <code>event_id</code> column is added when absent.
censor	Optional censoring indicator with one entry per volume: a logical vector, a 0/1 vector, or a vector of 1-based volume indices. Stored as the logical censor observation column that <code>fmridataset::temporal_schema()</code> recognises. <code>fmri_lm()</code> uses it without further configuration; it feeds AR estimation and whitening, and does not drop the censored volumes from the regression.
feature_ids	Optional stable feature IDs (one per column).
assay	Name of the assay holding datamat.

Value

An `fmri_frame` whose feature space is an `fmridataset::index_space()`.

See Also

[neurovec_frame\(\)](#), [nifti_frame\(\)](#), [latent_frame\(\)](#), [fmridataset::fmri_frame\(\)](#)

Examples

```
Y <- matrix(rnorm(80 * 3), 80, 3)
events <- data.frame(onset = c(5, 25, 45, 65),
                     condition = factor(c("A", "B", "A", "B")),
                     run = c(1, 1, 2, 2))
frame <- matrix_frame(Y, TR = 2, run_length = c(40, 40), event_table = events)
fmridataset::temporal_schema(frame)$run_lengths
```

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	<i>Meta-regression with ONE voxelwise covariate</i>
-------------------	---

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	<i>Extract Number of Subjects</i>
------------	-----------------------------------

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)
```

neurovec_frame

*Build an fmri_frame from in-memory NeuroVec objects***Description**

Extracts the masked time series of one or more neuroim2::NeuroVec runs and builds an `fmridataset::fmri_frame()` whose feature space is the `fmridataset::volume_space()` described by mask. Voxel time series are read with `neuroim2::series()`, so a SparseNeuroVec does not need to be densified.

Usage

```
neurovec_frame(
  scans,
  mask,
  TR,
  run_length = NULL,
  event_table = NULL,
  censor = NULL,
  assay = "bold"
)
```

Arguments

scans	A NeuroVec, or a list of them, one per run.
mask	A neuroim2::LogicalNeuroVol (or a NeuroVol whose non-zero voxels form the mask) on the same 3D grid as scans.
TR	Repetition time in seconds; one value, or one per run.
run_length	Volumes per run; defaults to the fourth dimension of each scan.
event_table, censor	As in <code>matrix_frame()</code> .
assay	Name of the assay.

Value

An fmri_frame over a volume_space.

See Also

`matrix_frame()`, `nifti_frame()`

nifti_frame

Build a lazy fmri_frame from NIfTI files

Description

Wraps `fmridataset::nifti_array_source()`, so headers and the mask are read at construction but no BOLD volumes are read until the fit needs them. For fMRIPrep derivatives see `fmridataset::read_bids_bold()`.

Usage

```
nifti_frame(
  scans,
  mask,
  TR,
  run_length = NULL,
  event_table = NULL,
  censor = NULL,
  base_path = ".",
  assay = "bold"
)
```

Arguments

scans	Character vector of NIfTI paths, one per run, all on the same grid.
mask	A NIfTI mask path, a <code>neuroim2::LogicalNeuroVol</code> , or a <code>fmridataset::volume_space()</code> .
TR	Repetition time in seconds; one value, or one per run.
run_length	Volumes per run; defaults to the fourth header dimension of each file.
event_table, censor	As in <code>matrix_frame()</code> .
base_path	Directory that relative scans and mask paths are resolved against.
assay	Name of the assay.

Value

An `fmri_frame` over the NIfTI source's `volume_space`.

See Also

`matrix_frame()`, `neurovec_frame()`, `fmridataset::nifti_array_source()`

noise_spec	<i>Temporal-noise specification</i>
------------	-------------------------------------

Description

Temporal-noise specification

Usage

```
noise_spec(
  struct = c("iid", "ar1", "ar2", "arp"),
  p = NULL,
  q = 0L,
  iter_gls = 1L,
  pooling = c("run", "global", "parcel"),
  shared_estimator = c("pooled_acvf", "mean_series"),
  parcels = NULL,
  voxelwise = FALSE,
  exact_first = FALSE,
  censor = NULL,
  shrink_c0 = 100L
)
```

Arguments

struct	One of iid, ar1, ar2, or arp.
p	AR order for struct = "arp".
q	Nonnegative moving-average order. Positive values request an ARMA(p, q) model, with the AR order determined by struct and p. The current MA-capable path supports runwise meta-estimation with run pooling, without censoring, parcel pooling, or voxelwise covariance estimation.
iter_gls	Maximum number of GLS refinement iterations. Standard design-corrected AR fitting uses the initial OLS residuals once and holds that estimate fixed for the GLS solve. Iterative refinement remains available to ARMA models (q > 0).
pooling	Temporal covariance pooling scope. With built-in shared AR estimation, "run" estimates one coefficient vector per run and "global" pools across runs.
shared_estimator	Spatial estimator for a shared temporal covariance. "pooled_acvf" (the default) pools residual autocovariances across voxels and therefore targets a typical voxel covariance. "mean_series" first averages residual values across voxels and targets the coherent spatial component; it is experimental, can be much more autocorrelated than an individual voxel. Because OLS projection is linear across response columns, the matching design correction remains valid after averaging.
parcels	Optional parcel labels for parcel pooling.

voxelwise	Estimate temporal covariance separately by voxel. In the built-in fitter this currently requires AR-only runwise meta-estimation, pooling = "run", iter_gls = 1, no censoring, and no volume weighting or soft-subspace projection. Robust fitting is supported, but robust AR re-estimation is not. Registered engines may define broader capabilities.
exact_first	Use exact first-observation AR scaling.
censor	Optional censor indices, logical mask, "auto", or "none". When left NULL a censor column carried by the dataset (as written by <code>matrix_frame()</code>) is used, which is what "auto" requests explicitly; "none" ignores such a column. Censoring feeds AR estimation and whitening only and does not drop volumes from the regression, so it has no effect when <code>struct = "iid"</code> .
shrink_c0	Parcel shrinkage constant used by supporting engines.

ols_t_cpp	<i>OLS t-test / ANCOVA across features</i>
-----------	--

Description

Subjects with a non-finite outcome for a feature are dropped from that feature only (complete-case analysis per feature); a feature with no more usable subjects than predictors, or whose remaining design is rank deficient, returns NaN.

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 (the residual degrees of freedom: a scalar when every feature uses all S subjects, otherwise a length-P vector), n_obs (length P, subjects used per feature), ok (P)

ols_t_vcov_cpp	<i>OLS with ONE voxelwise covariate</i>
----------------	---

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	<i>Extract P-values from a Model Fit</i>
----------	--

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("betas", "contrasts"), ...)
```

Arguments

- x The fitted model object
- ... Additional arguments passed to methods
- type For fmri_lm fits, the parameter family: "betas" (default) for the event coefficients returned by coef(), or "contrasts". "estimates" is accepted as a synonym of "betas".

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 <- matrix_frame(
  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

plot.fmri_model	<i>Plot the regressors of an fmri_model over time</i>
-----------------	---

Description

Draws the design-matrix columns as time series in stacked small multiples that share one time axis. Each event regressor gets its own row, coloured by condition (designs with more than 12 event columns get one row per event term instead). Baseline regressors are drawn in grey, one row per baseline term (drift, nuisance). Run-specific columns are drawn only within their own run, and no line is joined across a run boundary. Run intercepts are omitted, since a constant carries no temporal information.

Usage

```
## S3 method for class 'fmri_model'
plot(x, baseline = TRUE, ...)
```

Arguments

x	An fmri_model.
baseline	Logical; include baseline rows (default TRUE).
...	Unused.

Value

A ggplot2 object.

preflight

Preflight-check jobs before fan-out

Description

Validates one or more [fmri_jobs](#) against the structural contract their template implies, collecting issues per job. Intended to run on the driver before [run_jobs\(\)](#) / [export_jobs\(\)](#).

Usage

```
preflight(x, check_files = FALSE, on_issue = c("warn", "error", "collect"))
```

Arguments

<code>x</code>	A single fmri_job or a list of them.
<code>check_files</code>	Logical; for file-backed jobs, verify scan paths exist.
<code>on_issue</code>	One of "warn" (default), "error" (stop if any issue), or "collect" (return silently).

Details

Checks performed per job: template validity; every variable referenced by the design formula and block is a column of that job's event table; TR is positive; run lengths are consistent with the data (matrix_frame: rows match `sum(run_length)`; nifti_frame: one `run_length` per scan file); nuisance regressor rows match the total number of scans; and, when `check_files = TRUE`, that file-backed scans exist on disk.

The design-column check uses `all.vars()` on the formula, so it is deliberately conservative: a formula with a variable-valued HRF argument (e.g. `hrf(x, basis = my_basis)`) may flag `my_basis` as a missing column. Arguments of `feature()` are excluded, because those series live in the formula environment rather than the event table. Event tables supplied as file paths are not yet validated here.

Value

Invisibly, an object of class `fmri_preflight` with an `$issues` data frame (`job_id`, `message`) and `$ok`.

See Also

[validate_template\(\)](#), [run_jobs\(\)](#), [export_jobs\(\)](#)

Examples

```

tmpl <- fmri_template(onset ~ hrf(condition), ~ run)
job <- instantiate(tmpl, list(id = "sub-01",
                             scans = matrix(rnorm(80 * 2), 80, 2), TR = 2,
                             run_length = c(40, 40),
                             events = data.frame(onset = c(5, 45),
                                                  condition = factor(c("A", "B")),
                                                  run = c(1, 2))))
preflight(job)

```

print.fmri_meta	<i>Print Meta-Analysis Results</i>
-----------------	------------------------------------

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	<i>Print an fmri_lm_result object</i>
------------------	---------------------------------------

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	<i>Print method for fmri_ttest_fit</i>
----------------------	--

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	<i>Print Group Data Object</i>
------------------	--------------------------------

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	<i>Print Spatial FDR Results</i>
--------------------------	----------------------------------

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

projection_spec	<i>Nuisance-projection specification</i>
-----------------	--

Description

Nuisance-projection specification

Usage

```
projection_spec(  
  method = c("none", "soft_subspace"),  
  nuisance_mask = NULL,  
  nuisance_matrix = NULL,  
  lambda = "auto",  
  warn_redundant = TRUE  
)
```

Arguments

- method Projection rule.
- nuisance_mask Optional logical or numeric selector for nuisance features.
- nuisance_matrix Optional explicit nuisance-feature matrix.
- lambda Nonnegative regularization strength, "auto", or "gcv".
- warn_redundant Warn when projection overlaps modeled nuisance terms.

pvalues	<i>Compute P-values from Meta-Analysis</i>
---------	--

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	<i>Read All Data from HDF5 Files</i>
--------------	--------------------------------------

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_jobs	<i>Read jobs exported by export_jobs()</i>
-----------	--

Description

Read jobs exported by export_jobs()

Usage

```
read_jobs(dir)
```

Arguments

dir	Directory containing manifest.rds.
-----	------------------------------------

Value

The list of [fmri_jobs](#).

See Also

[export_jobs\(\)](#)

read_nifti_full	<i>Read All Data from NIfTI Files</i>
-----------------	---------------------------------------

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

realize_dataset	<i>Realize the dataset described by a job</i>
-----------------	---

Description

Reconstructs the fmri_frame from the job's [dataset_spec](#). For file-backed specs this is where data first becomes addressable (still lazily, through the frame's NIFTI array source).

Usage

```
realize_dataset(job)
```

Arguments

job	An fmri_job .
-----	-------------------------------

Value

An fmri_frame.

See Also

[build_model\(\)](#), [run_job\(\)](#)

reduce_betas	<i>Reducer: tidy beta-estimate table</i>
--------------	--

Description

Extracts the fitted condition estimates as a long data frame (one row per term x voxel), with standard error and a Wald statistic when available. Built on `coef()` / `standard_error()` for robustness.

Usage

```
reduce_betas(add_id = TRUE, include_baseline = FALSE)
```

Arguments

add_id	Logical; prepend a job_id column. Default TRUE.
include_baseline	Logical; include baseline/nuisance terms. Default FALSE.

Value

A reducer function suitable for `fmri_template()`.

See Also

Other reducers: `reduce_contrasts()`, `reduce_identity()`, `reduce_write_results()`

reduce_contrasts	<i>Reducer: tidy contrast table</i>
------------------	-------------------------------------

Description

Extracts the fitted contrasts as a long data frame (one row per contrast x voxel), optionally prefixed with the job id so results stack cleanly across subjects. Surfaces template-level contrasts (those passed to `fmri_template()`, applied by `run_job()`) when present, otherwise the model's formula-embedded contrasts via `coef()`. Returns a zero-row frame when no contrasts are defined.

Usage

```
reduce_contrasts(add_id = TRUE)
```

Arguments

add_id	Logical; prepend a job_id column. Default TRUE.
--------	---

Value

A reducer function suitable for `fmri_template()`.

See Also

Other reducers: [reduce_betas\(\)](#), [reduce_identity\(\)](#), [reduce_write_results\(\)](#)

reduce_identity	<i>Reducer: return the fitted model unchanged</i>
-----------------	---

Description

The escape hatch when you want the entire fmri_lm object per subject. Note this is the largest possible output.

Usage

```
reduce_identity()
```

Value

A reducer function suitable for [fmri_template\(\)](#).

See Also

Other reducers: [reduce_betas\(\)](#), [reduce_contrasts\(\)](#), [reduce_write_results\(\)](#)

reduce_write_results	<i>Reducer: write per-subject results to disk (BIDS-keyed)</i>
----------------------	--

Description

Calls [write_results\(\)](#) on the fitted model, keying the output filenames by the job's meta (subject, task, space). Returns the vector of created file paths. This is the default reducer for large studies: workers write compact statistical maps that [group_data\(\)](#) / [fmri_meta\(\)](#) can pick up for the group level.

Usage

```
reduce_write_results(
  format = c("h5", "nifti", "gds"),
  stats = c("beta", "tstat"),
  contrasts = NULL,
  path = NULL,
  desc = "GLM",
  overwrite = FALSE
)
```

Arguments

format	Output format passed to <code>write_results()</code> .
stats	Which contrast statistics to write (mapped to <code>contrast_stats</code>).
contrasts	Optional restriction to specific contrasts.
path	Output directory (passed to <code>write_results()</code>).
desc	BIDS desc- label.
overwrite	Overwrite existing files.

Value

A reducer function suitable for `fmri_template()`.

Note

`write_results()` requires BIDS entities: each job’s meta should carry subject (defaults to the job id), task, and space. A missing task/space surfaces as an error when the reducer runs.

See Also

Other reducers: `reduce_betas()`, `reduce_contrasts()`, `reduce_identity()`

register_basis	<i>Register an HRF/basis constructor</i>
----------------	--

Description

Register an HRF/basis constructor

Usage

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

Arguments

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

Value

Invisibly, TRUE.

register_engine	<i>Register a plugin engine for fmri_lm</i>
-----------------	---

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 <code>fit(model, dataset, args, cfg)</code> and expected to return an <code>fmri_lm</code> object.
preflight	Optional function invoked before fitting; receives the same arguments as <code>fit</code> and can signal errors early.
capabilities	Optional named list describing engine support for global <code>fmri_lm()</code> options. Recognized fields currently include <code>robust</code> , <code>preprocessing</code> , <code>ma</code> , <code>ar_voxelwise</code> , <code>ar_by_cluster</code> , plus contextual rules such as <code>requires_event_regressors</code> , <code>requires_parcel_for_by_cluster</code> , and <code>forbid_by_cluster_dataset_classes</code> (class names matched against the dataset or against its <code>fmridataset::space()</code> , e.g. "basis_space" for latent frames).

Value

Invisibly, TRUE.

register_run_backend	<i>Register an execution backend for run_jobs()</i>
----------------------	---

Description

Adds a named backend that `run_jobs()` can dispatch to. This is how to plug in a custom scheduler driver without modifying **fmrireg**. The builtins "sequential" and "future" are always available.

Usage

```
register_run_backend(name, fn, overwrite = FALSE)
```

Arguments

name	Backend name (a string).
fn	A function function(jobs, run_one, ...) returning an order-preserving list of per-job results.
overwrite	Allow replacing an existing backend of the same name.

Value

The backend name, invisibly.

See Also

[run_jobs\(\)](#), [run_backends\(\)](#)

Examples

```
register_run_backend("first_only", function(jobs, run_one, ...) {
  list(run_one(jobs[[1]]))
})
```

resolve_basis	<i>Resolve a registered basis function by name</i>
---------------	--

Description

This function is used internally by the formula processing system to lazily resolve basis names (e.g., "cca2", "cca3", "spmgl") 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)
```

robust_spec	<i>Robust-estimation specification</i>
-------------	--

Description

Robust-estimation specification

Usage

```
robust_spec(  
  type = c("none", "huber", "bisquare"),  
  k_huber = 1.345,  
  c_tukey = 4.685,  
  max_iter = 2L,  
  scale_scope = c("run", "global", "voxel"),  
  reestimate_phi = FALSE  
)
```

Arguments

type	Robust loss, or "none" for ordinary least squares.
k_huber	Tuning constant for the Huber loss.
c_tukey	Tuning constant for Tukey's bisquare loss.
max_iter	Maximum number of robust reweighting iterations.
scale_scope	Scope at which robust scale is estimated.
reestimate_phi	Re-estimate temporal-noise parameters after robust reweighting.

run_backends	<i>List registered execution backends</i>
--------------	---

Description

List registered execution backends

Usage

```
run_backends()
```

Value

A character vector of backend names.

See Also

[register_run_backend\(\)](#), [run_jobs\(\)](#)

run_job	<i>Run a single job</i>
---------	-------------------------

Description

Realizes the dataset, assembles the model, fits it with the template's control options (and engine, if any), and applies the template's reducer.

Usage

```
run_job(job, progress = FALSE)
```

Arguments

job	An fmri_job .
progress	Logical; show a fitting progress bar.

Value

The reducer's output, or the fitted `fmri_lm` if the template has no reducer.

See Also

[run_jobs\(\)](#), [build_model\(\)](#)

run_jobs	<i>Run a batch of jobs</i>
----------	----------------------------

Description

Executes a list of [fmri_jobs](#), returning an `fmri_batch_result`. By default each job is isolated: a failure is captured rather than aborting the batch. Set `parallel = TRUE` to dispatch through the active `future::plan()` (e.g. `multisession`, `cluster`, or a `future.batchtools` cluster backend).

Usage

```
run_jobs(
  jobs,
  parallel = FALSE,
  progress = FALSE,
  on_error = c("isolate", "stop"),
  backend = NULL
)
```

Arguments

jobs	A single fmri_job or a list of them.
parallel	Logical shorthand; TRUE selects the "future" backend, FALSE the "sequential" one. Ignored if backend is given.
progress	Logical; per-job fitting progress.
on_error	Either "isolate" (default; capture per-job errors) or "stop" (fail the batch on the first error).
backend	Optional execution backend name (see run_backends()). Overrides parallel. The "future" backend honours the active <code>future::plan()</code> (including <code>future.batchtools</code> cluster plans).

Value

An object of class `fmri_batch_result`.

See Also

[run_job\(\)](#), [batch_values\(\)](#), [batch_errors\(\)](#), [register_run_backend\(\)](#)

Examples

```
## Not run:
jobs <- instantiate(tmpl, manifest)
future::plan(future::multisession, workers = 4)
res <- run_jobs(jobs, parallel = TRUE)
values <- batch_values(res)

## End(Not run)
```

se	<i>Extract Standard Errors from Meta-Analysis</i>
----	---

Description

Extract Standard Errors from Meta-Analysis

Usage

`se(object)`

Arguments

object	An <code>fmri_meta</code> 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)
```

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 controlled event-related teaching 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	<i>Simulate fMRI Time Courses, Return Shared Onsets + Column-Specific Amplitudes/Durations</i>
----------------------	--

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 `fmri_frame` 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. <code>fmrihrf::HRF_SPMG1</code> .
n_events	Number of events (ignored if onsets is provided).
onsets	Optional non-empty numeric vector of finite event onsets in $[0, \text{total_time} - \text{buffer})$. Explicit values and their order are preserved, determine n_events, and skip ISI generation.
isi_dist	One of "even", "uniform", or "exponential". Default is "even" so events are evenly spaced within $\text{total_time} - \text{buffer}$.
isi_min, isi_max	For <code>isi_dist="uniform"</code> .
isi_rate	For <code>isi_dist="exponential"</code> .
durations	Numeric, scalar or length-n_events. If <code>duration_sd>0</code> , 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 <code>amplitude_sd>0</code> , 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 fmri_frame'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 fmri_frame 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 structured fMRI-like teaching 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 a declared structured-noise model. 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	Finite positive signal-to-noise ratio, defined as the empirical standard deviation of the clean condition signals divided by the empirical standard deviation of the complete structured-noise matrix (default is 0.5)
hrf	Hemodynamic response function to use (default is fmrihrf::HRF_SPMG1)
seed	Optional seed for reproducibility (default is NULL). Reusing a seed with a different snr preserves the clean signal and standardized noise realization while changing only the noise scale.

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
old_par <- par(no.readonly = TRUE)
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")
par(old_par)
```

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

<code>enabled</code>	Logical. Whether to apply soft subspace projection.
<code>nuisance_mask</code>	Path to NIfTI mask or logical vector indicating nuisance voxels.
<code>nuisance_matrix</code>	Pre-computed nuisance timeseries matrix (alternative to mask).
<code>lambda</code>	Ridge penalty: numeric, "auto", or "gcv".
<code>warn_redundant</code>	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 $\times$ 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 :

- $Y_{\text{clean}} = P_{\lambda} Y$
- $X_{\text{clean}} = P_{\lambda} X$ (important to avoid bias)

Lambda Selection

The shrinkage parameter λ controls aggressiveness:

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

Three selection methods are available:

"auto" Singular value heuristic: $\lambda = \text{median}(d^2)$ 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 $\text{RSS}/(1-\text{df}/n)^2$ for ridge regression of Y on N . Finds λ 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 λ 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, character or factor vector of group IDs for each feature (e.g., parcel IDs, block IDs). Must have same length as z or p. Groups are numbered 1..G by sorted unique value, or by factor level (unused levels included) when group is a factor. This numbering indexes neighbors, pi0_raw, pi0_smooth and m_g. Features with NA group are excluded from π_0 estimation and receive weight 1 before normalisation.
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 element g is an integer vector of 1-based neighbor IDs of group g, both in the group numbering described under group. 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 the z-values (default: TRUE). See Details.
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 (π_0) within spatial groups
2. Optionally smooths these estimates across neighboring groups
3. Uses the π_0 estimates to weight the Benjamini-Hochberg procedure
4. Provides more power in regions with true signal while maintaining FDR control

Weights are $1/\pi_0$ of each feature's group, normalised to sum to the number of finite p-values, and the weighted BH procedure of Genovese, Roeder & Wasserman (2006) is applied: reject $p_i \leq w_i \alpha k / n$.

Empirical null. When z is supplied with `empirical_null = TRUE`, μ_0 and σ_0 are estimated by Efron's (2004) central matching: a quadratic is fitted by Poisson regression to the log histogram counts of z within ± 1.5 robust SDs of the median. Because a minority of non-null features only changes the intercept of the log density near the centre and not its curvature, this estimate is consistent under $N(0, 1)$ and stays close to it when, for example, 10% of features are true signals. With fewer than 500 finite z-values (or if the fit fails) the median and normal-consistent MAD of all finite z-values are used instead; with fewer than 10 the theoretical null is used.

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 <code>empirical_null = TRUE</code>)
sigma0	Numeric scalar; estimated null SD (if <code>empirical_null = TRUE</code>)
groups	Integer vector of group IDs (1..G) in the numbering described under group

G	Integer scalar; number of groups
m_g	Integer vector; number of features per group
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).
- Genovese, Roeder & Wasserman (2006). False discovery control with p-value weighting.
- Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis.

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	<i>Extract Standard Errors from a Model Fit</i>
----------------	---

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("betas", "contrasts"), recon = FALSE, ...)

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

Arguments

<code>x</code>	The fitted model object
<code>...</code>	Additional arguments passed to methods
<code>type</code>	For <code>fmri_lm</code> fits, the parameter family: "betas" (default) for the event coefficients returned by <code>coef()</code> , or "contrasts". "estimates" is accepted as a synonym of "betas".
<code>recon</code>	Logical; whether to reconstruct the full matrix representation (default: FALSE)

Value

A tibble or matrix containing standard errors of parameter estimates. For `fmri_lm` fits, one row per voxel and one column per term, in the same orientation as `coef()`.

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 <- matrix_frame(
  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("betas", "contrasts", "F"), ...)
```

Arguments

x	The fitted model object
...	Additional arguments passed to methods
type	For <code>fmri_lm</code> fits, the parameter family whose test statistics are returned: "betas" (default) gives the t-statistics of the event coefficients, "contrasts" the t-statistics of the simple contrasts, and "F" the F-statistics of the F-contrasts. <code>stats()</code> never returns the estimates themselves; use <code>coef()</code> for those. "estimates" is a deprecated synonym of "betas" (it reads as "the estimates" but returns t-statistics) and warns; set <code>options(fmrireg.suppress_deprecation = TRUE)</code> to silence it.

Value

A tibble or matrix containing test statistics. For `fmri_lm` fits, a tibble with one row per voxel and one column per term, in the same orientation as `coef()`.

See Also

[coef\(\)](#) for the estimates.

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 <- matrix_frame(
  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	<i>Summary of Group Data Object</i>
--------------------	-------------------------------------

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	<i>Summary of Spatial FDR Results</i>
----------------------------	---------------------------------------

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

<code>t_to_beta_se</code>	<i>Convert t-statistics to Effect Sizes</i>
---------------------------	---

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

<code>t_to_d</code>	<i>Convert T-statistics to Effect Sizes and Variances</i>
---------------------	---

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)
```

 theme_fmrireg

The fmrireg ggplot2 theme

Description

A restrained theme used by all fmrireg plots: hairline major grid only, left-aligned titles, and muted axis ink so the data carry the contrast. Add it to any ggplot, or override it by adding your own theme afterwards.

Usage

```
theme_fmrireg(
  base_size = 11,
  base_family = "",
  grid = c("y", "x", "xy", "none")
)
```

Arguments

base_size	Base font size in points.
base_family	Base font family.
grid	Which major gridlines to keep: "y", "x", "xy", or "none".

Details

fmrireg plots use this theme by default. To use a different base theme for every fmrireg plot, set `options(fmrireg.plot_theme = <theme>)`, for example `options(fmrireg.plot_theme = ggplot2::theme_bw())`.

Value

A `ggplot2::theme()` object.

Examples

```
library(ggplot2)
ggplot(mtcars, aes(wt, mpg)) + geom_point() + theme_fmrireg()
```

tidy	<i>Tidy generic</i>
------	---------------------

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_hrf_estimate
<i>Tidy a smooth HRF estimate</i>

Description

Tidy a smooth HRF estimate

Usage

```
## S3 method for class 'fmri_hrf_estimate'  
tidy(x, curve = NULL, voxel = NULL, ...)
```

Arguments

x	An fmri_hrf_estimate object.
curve	Optional curve names to retain.
voxel	Optional voxel names or indices to retain.
...	Unused.

Value

A tibble with one row per time, curve, and voxel.

tidy.fmri_meta	<i>Tidy Meta-Analysis Results</i>
----------------	-----------------------------------

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	<i>Tidy fitted HRF curves from an fmri_lm fit</i>
-----------------	---

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 <code>fmri_lm</code> object.
sample_at	Numeric vector of time points passed to <code>fitted_hrf()</code> .
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 <code>fitted_hrf()</code> .

Value

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

validate_job	<i>Validate an fmri_job</i>
--------------	-----------------------------

Description

Validate an `fmri_job`

Usage

```
validate_job(x)
```

Arguments

x	An <code>fmri_job</code> .
---	----------------------------

Value

TRUE invisibly if valid; otherwise an error is raised.

validate_template	<i>Validate an fmri_template</i>
-------------------	----------------------------------

Description

Re-checks the structural invariants of a template. Useful in preflight before fanning a model out over many subjects.

Usage

validate_template(x)

Arguments

x An [fmri_template](#).

Value

TRUE invisibly if valid; otherwise an error is raised.

variance_model	<i>Extract the structured variance model</i>
----------------	--

Description

Extract the structured variance model

Usage

variance_model(x)

Arguments

x A fitted fmri_lm object.

Value

An fmri_lm_variance_model object.

variance_spec	<i>Variance and reference-distribution specification</i>
---------------	--

Description

Variance and reference-distribution specification

Usage

```
variance_spec(  
  method = c("model", "hac", "sandwich"),  
  max_lag = "auto",  
  taper = c("none", "tukey", "parzen"),  
  debias = TRUE,  
  df = c("residual", "satterthwaite")  
)
```

Arguments

method	Covariance estimator: model-based, HAC, or sandwich.
max_lag	Maximum HAC lag, or "auto" for a data-dependent lag.
taper	HAC lag-window taper.
debias	Apply the estimator's finite-sample correction.
df	Reference degrees-of-freedom method.

volume_quality	<i>Volume Quality Metrics and Temporal Weighting</i>
----------------	--

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 $|u| \leq 1$

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	<i>Compute Volume Quality Weights from Data</i>
----------------	---

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

<code>Y</code>	Numeric matrix of fMRI data (time x voxels).
<code>method</code>	Weighting method passed to <code>dvars_to_weights</code> .
<code>threshold</code>	Threshold passed to <code>dvars_to_weights</code> .
<code>return_dvars</code>	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")
```

weights_spec	<i>Volume-weight specification</i>
--------------	------------------------------------

Description

Volume-weight specification

Usage

```
weights_spec(  
  method = c("none", "inverse_squared", "soft_threshold", "tukey"),  
  threshold = 1.5,  
  values = NULL  
)
```

Arguments

method	Volume-weight construction rule.
threshold	Positive threshold or tuning constant for the selected rule.
values	Optional explicit finite, nonnegative volume weights. Supplying values enables volume weighting even when method = "none".

welch_t_cpp	<i>Welch two-sample t-test across features</i>
-------------	--

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	<i>Write Results from fMRI Analysis</i>
---------------	---

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	<i>Write Results from fMRI Linear Model</i>
-----------------------	---

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", "nifti", "gds"),
  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	BIDS desc- label of the analysis (default: "GLM"), sanitised to letters and digits. Beta files are labelled desc-<desc>; statistic maps desc-<desc><stat> (e.g. desc-run1tstat), except that the default "GLM" is omitted (desc-tstat), the naming collect_results() reads by default.
format	Output format(s). One or more of "h5" (BIDS-style HDF5 outputs), "nifti" (BIDS-style NIFTI outputs, written as .nii.gz), and "gds" (fmrigds-compatible assays plus an .rds plan). Pass a character vector to write multiple formats at once (e.g. c("h5", "nifti")). Defaults to "h5" only; "h5" requires the suggested package fmristore, "nifti" does not. NIFTI maps are stored as 32-bit floats (datatype FLOAT32), so their values are the fitted values rounded to single precision.
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 fmri_lm 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. NIfTI maps are stored as 32-bit floats (datatype FLOAT32).
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	<i>Back-transform Fisher's Z to Correlation</i>
--------	---

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

- * **datasets**
 - .fmrireg_engine_registry, 6
- * **estimate_betas**
 - estimate_betas, 39
- * **hrf**
 - fitted_hrf, 52
- * **model_estimation**
 - estimate_betas, 39
- * **reducers**
 - reduce_betas, 116
 - reduce_contrasts, 116
 - reduce_identity, 117
 - reduce_write_results, 117
- * **result_export**
 - write_results, 153
- * **statistical_measures**
 - coef_names, 20
 - p_values, 105
 - standard_error, 136
 - stats, 138
- .fmrireg_engine_registry, 6
- .resample_param, 6
- apply_soft_projection, 7
- ar_parameters, 7
- as.array.NeuroVec, 8
- as.matrix.fmri_hrf_estimate
 - (fmri_hrf_estimate-methods), 55
- as_manifest, 9
- as_manifest(), 75
- autoplot.fmri_lm, 9
- autoplot.Reg, 12
- baseline_model, 41
- baseline_model(), 13, 43
- baseline_spec, 13, 15
- baseline_spec(), 70, 71
- batch_errors, 14
- batch_errors(), 14, 123
- batch_values, 14
- batch_values(), 14, 123
- build_model, 15
- build_model(), 115, 122
- coef, 21
- coef(), 10, 116, 138
- coef.fmri_hrf_estimate
 - (fmri_hrf_estimate-methods), 55
- coef.fmri_lm, 15
- coef.fmri_meta, 17
- coef_image, 16, 19–21
- coef_image(coef_image.fmri_lm), 17
- coef_image.fmri_lm, 17
- coef_image.fmri_ttest_fit, 18
- coef_images(coef_images.fmri_lm), 19
- coef_images.fmri_lm, 19
- coef_names, 16, 19, 20, 20, 106, 137, 138
- collect_results, 21
- collect_results(), 154
- compute_dvars, 22
- compute_lm_contrasts, 23
- compute_lm_contrasts_from_suffstats,
 - 24
- compute_spec, 26
- compute_spec(), 58, 59, 70
- contrast, 26
- contrast.fmri_meta, 27
- contrast_set, 28
- correlation_map.fmri_model, 28
- create_3d_blocks, 29, 136
- create_design_matrix_from_benchmark,
 - 30
- dataset_spec, 31, 56, 115
- design_map.fmri_model, 32
- design_matrix.fmri_lm, 33
- design_matrix.fmri_model, 33
- design_plot, 34
- dvars_to_weights, 36

engine_spec, 37
 engine_specs, 38
 estimate, 38
 estimate_betas, 39, 80, 82
 estimate_hrf, 42
 estimation_spec, 44
 estimation_spec(), 60
 evaluate_method_performance, 44
 event_model, 41
 event_model(), 43
 export_jobs, 45
 export_jobs(), 56, 108, 114
 extract_csv_data, 46
 extract_nuisance_timeseries, 47

 fit_contrasts, 47
 fit_contrasts.default, 48
 fit_contrasts.fmri_lm, 49
 fit_glm_from_suffstats, 49
 fit_glm_on_transformed_series, 50
 fit_glm_with_config, 51
 fit_glm_with_config(), 50
 fitted_hrf, 52
 fitted_hrf(), 43, 146, 147
 fitted_hrf.fmri_lm, 53
 flip_sign, 54
 fmri_benchmark_datasets, 54
 fmri_hrf_estimate-methods, 55
 fmri_job, 15, 31, 45, 56, 70, 90, 108, 114, 115, 122, 123, 147
 fmri_job(), 32, 90
 fmri_latent_lm, 57
 fmri_lm, 58
 fmri_lm(), 52, 59, 96
 fmri_lm_control, 59
 fmri_lm_control(), 58, 59, 70, 71
 fmri_meta, 60, 64, 72, 82, 97, 113, 143, 157
 fmri_meta(), 21, 22, 117
 fmri_meta_fit, 63
 fmri_meta_fit_contrasts, 64
 fmri_meta_fit_cov, 65
 fmri_meta_fit_extended, 66
 fmri_model, 67
 fmri_ols_fit, 67
 fmri_rlm, 68
 fmri_template, 56, 70, 90, 148
 fmri_template(), 13, 116–118
 fmri_ttest, 71
 fmridataset::basis_space(), 91
 fmridataset::basis_space_from_decoder(), 91
 fmridataset::basis_synthesis(), 91
 fmridataset::event_table(), 96
 fmridataset::fmri_frame(), 96, 97, 101
 fmridataset::index_space(), 96
 fmridataset::nifti_array_source(), 102
 fmridataset::read_bids_bold(), 102
 fmridataset::temporal_schema(), 96
 fmridataset::volume_space(), 101, 102
 fmrihrf::HRF_SPMG1(), 52
 fmrireg_cli, 73
 from_bids, 74
 from_bids(), 9, 90

 generate_interaction_contrast, 75
 generate_main_effect_contrast
 (generate_interaction_contrast), 75
 geom_raster, 33
 geom_tile, 29
 get_benchmark_summary, 76
 get_contrasts, 77
 get_covariates, 77
 get_rois, 78
 get_subjects, 78
 ggplot2::theme(), 144
 glm_lss, 79, 82
 glm_ols, 80, 81
 group_data, 61, 82
 group_data(), 21, 117
 group_data_from_csv, 83
 group_data_from_fmri_lm, 85
 group_data_from_h5, 86
 group_data_from_nifti, 87

 hrf_smoothing_kernel, 88

 install_cli, 89
 instantiate, 90
 instantiate(), 9, 45, 46, 56, 70, 71, 74, 75

 latent_frame, 41, 91
 latent_frame(), 39, 57, 97
 list_benchmark_datasets, 92
 load_benchmark_dataset, 92
 lowrank_control, 94
 lowrank_control(), 59

 ma_parameters, 95

- matrix_frame, [41](#), [96](#)
- matrix_frame(), [32](#), [39](#), [79](#), [81](#), [91](#), [101](#), [102](#), [104](#)
- meta_effective_n, [97](#)
- meta_fit_vcov_cpp, [98](#)
- mixed_solve_cpp, [99](#)
- n_subjects, [100](#)
- neurovec_frame, [41](#), [101](#)
- neurovec_frame(), [39](#), [97](#), [102](#)
- nifti_frame, [102](#)
- nifti_frame(), [32](#), [39](#), [97](#), [101](#)
- noise_spec, [103](#)
- noise_spec(), [60](#)
- ols_t_cpp, [104](#)
- ols_t_vcov_cpp, [105](#)
- p_values, [15](#), [16](#), [21](#), [105](#), [137](#), [138](#)
- paired_diff_block, [106](#)
- plot.fmri_lm(autoplot.fmri_lm), [9](#)
- plot.fmri_model, [107](#)
- predict.fmri_hrf_estimate
(fmri_hrf_estimate-methods), [55](#)
- preflight, [108](#)
- preflight(), [75](#)
- print.fmri_hrf_estimate
(fmri_hrf_estimate-methods), [55](#)
- print.fmri_lm(print.fmri_model), [110](#)
- print.fmri_meta, [109](#)
- print.fmri_model, [110](#)
- print.fmri_rlm(print.fmri_model), [110](#)
- print.fmri_ttest_fit, [110](#)
- print.group_data, [111](#)
- print.spatial_fdr_result, [111](#)
- projection_spec, [112](#)
- projection_spec(), [60](#)
- pvalues, [112](#)
- r_to_z, [113](#)
- read_h5_full, [114](#)
- read_jobs, [114](#)
- read_jobs(), [46](#)
- read_nifti_full, [115](#)
- realize_dataset, [115](#)
- realize_dataset(), [15](#)
- reduce_betas, [116](#), [117](#), [118](#)
- reduce_contrasts, [116](#), [116–118](#)
- reduce_identity, [116](#), [117](#), [117](#), [118](#)
- reduce_write_results, [116](#), [117](#), [117](#)
- reduce_write_results(), [21](#), [22](#), [70](#)
- register_basis, [118](#)
- register_engine, [119](#)
- register_engine(), [70](#)
- register_run_backend, [119](#)
- register_run_backend(), [121](#), [123](#)
- resolve_basis, [120](#)
- robust_spec, [121](#)
- robust_spec(), [60](#)
- run_backends, [121](#)
- run_backends(), [120](#), [123](#)
- run_job, [122](#)
- run_job(), [15](#), [56](#), [90](#), [115](#), [116](#), [123](#)
- run_jobs, [122](#)
- run_jobs(), [14](#), [22](#), [46](#), [108](#), [119–122](#)
- se, [123](#)
- simulate_bold_signal, [124](#)
- simulate_fmri_matrix, [125](#)
- simulate_noise_vector, [127](#)
- simulate_simple_dataset, [128](#)
- soft_projection, [130](#)
- soft_subspace_options, [131](#)
- soft_subspace_projection, [132](#)
- spatial_fdr, [134](#)
- standard_error, [15](#), [16](#), [21](#), [106](#), [136](#), [138](#)
- standard_error(), [116](#)
- stats, [10](#), [15](#), [16](#), [21](#), [106](#), [137](#), [138](#)
- summary.fmri_meta, [139](#)
- summary.fmri_ttest_fit, [140](#)
- summary.group_data, [141](#)
- summary.spatial_fdr_result, [141](#)
- t_to_beta_se, [142](#)
- t_to_d, [142](#)
- term_matrices.fmri_model, [143](#)
- theme_fmrireg, [144](#)
- tidy, [145](#)
- tidy.fmri_hrf_estimate, [145](#)
- tidy.fmri_meta, [146](#)
- tidy_fitted_hrf, [146](#)
- validate_job, [147](#)
- validate_template, [148](#)
- validate_template(), [108](#)
- variance_model, [148](#)
- variance_spec, [149](#)
- variance_spec(), [60](#)

volume_quality, [149](#)
volume_weights, [151](#)

weights_spec, [152](#)
weights_spec(), [60](#)
welch_t_cpp, [152](#)
write_results, [153](#)
write_results(), [117](#), [118](#)
write_results.fmri_lm, [86](#), [153](#), [155](#)
write_results.fmri_meta, [155](#)

z_to_r, [156](#)
zscores, [157](#)