

Package: fmriLSS (via r-universe)

October 10, 2026

Type Package

Title Least Squares Separate (LSS) Analysis for fMRI Data

Version 0.2.0

Description Implements efficient least squares separate (LSS) analysis for functional magnetic resonance imaging (fMRI) data, following Mumford et al. (2012) [<doi:10.1016/j.neuroimage.2011.08.076>](https://doi.org/10.1016/j.neuroimage.2011.08.076). LSS estimates trial-by-trial activation patterns in event-related designs. The package provides reference R implementations, optimized C++ backends, prewhitening, and methods for voxel-specific haemodynamic response functions, including a fast implementation of GLMsingle (Prince et al., 2022) [<doi:10.7554/eLife.77599>](https://doi.org/10.7554/eLife.77599).

License GPL-3

Copyright The vendored GLMsingle HRF library and ported GLMsingle and fraccirc algorithms are by their authors; see inst/COPYRIGHTS.

Encoding UTF-8

Depends R (>= 4.0.0)

Imports MASS, Rcpp (>= 0.12.0), fmrihrf (>= 0.3.0), fmriAR (>= 0.3.3), bigmemory, ggplot2, glmnet

LinkingTo Rcpp, RcppArmadillo, roptim, bigmemory, BH

Suggests spelling, testthat (>= 3.0.0), knitr, rmarkdown, pkgdown, progress, fmridesign, jsonlite, Matrix, albersdown (>= 2.1.0), ragg, systemfonts

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

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/testthat/edition 3

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

BugReports <https://github.com/bbuchtsbaum/fmrlss/issues>

Language en-US

Config/Needs/website albersdown

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

Date/Publication 2026-10-10 20:38:15 UTC

RemoteUrl <https://github.com/bbuchtsbaum/fmrlss>

RemoteRef HEAD

RemoteSha 188d0dd9085bd1d5acdddf7bffd87920f132a9f6

Contents

benchmark_mixed_solve	3
calculate_recovery_metrics	4
coef.glmsingle_fit	5
compare_hrf_recovery	5
create_lwu_grid	6
estimate_voxel_hrf	7
fit_oasis_grid	8
fmrlss_options	9
generate_lwu_data	10
generate_rapid_design	11
glmsingle	12
glmsingle_design	16
glmsingle_hrf_library	17
item_build_design	18
item_compute_u	19
item_cv	20
item_fit	22
item_from_lsa	23
item_predict	24
item_slice_fold	25
lsa	26
lss	27
lss_beta_cpp	33
lss_cpp_optimized	34
lss_design	35
lss_fit_wrappers	38
lss_naive	39
lss_naive_fit	40
lss_optimized	41
lss_optimized_fit	42
lss_rank1	43
lss_sbhm	46
lss_sbhm_design	49
lss_with_hrf	50

LSSBeta	52
mixed_precompute	53
mixed_solve	53
mixed_solve_optimized	55
oasis_options	56
plot_hrf_comparison	57
prewhiten_options	58
project_confounds	59
project_confounds_cpp	60
sbhm_build	61
sbhm_hrf	63
sbhm_match	64
sbhm_prepass	65
sbhm_project	66
stglmnet_options	67
VoxelHRF	69
Index	70

benchmark_mixed_solve	<i>Benchmark Mixed Model Implementations</i>
-----------------------	--

Description

Compare performance between the standard mixed_solve implementation and the optimized mixed_solve_optimized version.

Usage

benchmark_mixed_solve(X, Z, K = NULL, Y, n_reps = 5)

Arguments

- | | |
|--------|---|
| X | Fixed effects design matrix |
| Z | Random effects design matrix |
| K | Kinship matrix (optional, defaults to identity) |
| Y | Response matrix (n x V) |
| n_reps | Number of repetitions for benchmarking |

Value

Data frame with timing results

Examples

```
X <- matrix(rnorm(100 * 2), 100, 2)
Z <- matrix(rnorm(100 * 3), 100, 3)
Y <- matrix(rnorm(100 * 5), 100, 5)
benchmark_mixed_solve(X, Z, Y = Y, n_reps = 2)
```

calculate_recovery_metrics

Calculate HRF Recovery Metrics

Description

Evaluates how well each method recovered the true HRF

Usage

```
calculate_recovery_metrics(results, true_hrf)
```

Arguments

results	Output from compare_hrf_recovery
true_hrf	Ground truth HRF sampled on results\$hrf_times. Shape comparisons normalize each curve to unit maximum absolute height.

Value

Data frame with recovery metrics

Examples

```
onsets <- generate_rapid_design(n_events = 4, total_time = 60, seed = 1)
sim <- generate_lwu_data(onsets, total_time = 60, n_voxels = 2, seed = 1)
grid <- create_lwu_grid(n_tau = 2, n_sigma = 2, n_rho = 2)
res <- compare_hrf_recovery(sim, hrf_grid = grid)
calculate_recovery_metrics(res, sim$true_hrf)
```

coef.glmsingle_fit	<i>Single-trial betas from a glmsingle fit</i>
--------------------	--

Description

Single-trial betas from a glmsingle fit

Usage

```
## S3 method for class 'glmsingle_fit'
coef(object, type = c("d", "c", "b", "a"), ...)
```

Arguments

object	A glmsingle_fit.
type	Model type: "d" (fractional ridge, default), "c" (GLMdenoise), "b" (HRF library) or "a" (ON-OFF, one value per voxel).
...	Unused.

Value

Trials x voxels matrix (a vector for type "a").

Examples

```
example("glmsingle", echo = FALSE)
beta <- coef(fit)
dim(beta)
```

compare_hrf_recovery	<i>Compare HRF Recovery Methods</i>
----------------------	-------------------------------------

Description

Compares OASIS, SPMG1, SPMG3, and FIR for HRF recovery

Usage

```
compare_hrf_recovery(data, hrf_grid = NULL)
```

Arguments

data	Synthetic data from generate_lwu_data
hrf_grid	Optional pre-computed HRF grid for OASIS

Value

List with results from all methods

Examples

```
onsets <- generate_rapid_design(n_events = 4, total_time = 60, seed = 1)
sim <- generate_lwu_data(onsets, total_time = 60, n_voxels = 2, seed = 1)
grid <- create_lwu_grid(n_tau = 2, n_sigma = 2, n_rho = 2)
res <- compare_hrf_recovery(sim, hrf_grid = grid)
names(res)
```

create_lwu_grid	<i>Create LWU HRF Grid for OASIS Search</i>
-----------------	---

Description

Generates a grid of LWU HRF models with varying parameters

Usage

```
create_lwu_grid(
  tau_range = c(4, 8),
  sigma_range = c(1.5, 3.5),
  rho_range = c(0.1, 0.6),
  n_tau = 5,
  n_sigma = 3,
  n_rho = 3
)
```

Arguments

tau_range	Range of tau values to test
sigma_range	Range of sigma values to test
rho_range	Range of rho values to test
n_tau	Number of tau values in grid
n_sigma	Number of sigma values in grid
n_rho	Number of rho values in grid

Value

An object of class "fmrilss_lwu_grid": a list containing the HRF functions in hrfs and their parameter data frame in parameters. The object can be supplied directly as library_spec\$pgrid to [sbhm_build\(\)](#).

Examples

```
grid <- create_lwu_grid(n_tau = 2, n_sigma = 2, n_rho = 2)
nrow(grid$parameters)
```

estimate_voxel_hrf	<i>Estimate Voxel-wise HRF Basis Coefficients</i>
--------------------	---

Description

Fits a common-amplitude GLM to estimate one pooled HRF shape for every voxel. The HRF basis and response are residualized against the complete fixed and nuisance span before fitting. Estimation fails when that span makes the HRF basis unidentifiable or rank deficient.

Usage

```
estimate_voxel_hrf(
  Y,
  events,
  basis,
  nuisance_regs = NULL,
  sframe = NULL,
  fixed_regs = NULL,
  ref_hrf = NULL
)
```

Arguments

Y	Numeric matrix of BOLD data (time x voxels).
events	Data frame with onset, duration and condition columns. For a multi-run sampling frame, a run column is required and onsets are physical times relative to that run.
basis	HRF object from the fmrihrf package.
nuisance_regs	Optional finite numeric matrix of nuisance regressors.
sframe	Explicit fmrihrf sampling frame defining scan times and TR. Required; onset and duration values use its physical-time units.
fixed_regs	Optional finite numeric matrix of fixed/common regressors. An intercept is added when it is not already in their span.
ref_hrf	HRF used to orient the sign of each estimated shape. Defaults to the canonical fmrihrf::HRF_SPMG1.

Value

A **VoxelHRF** object containing at least:

coefficients	Matrix of positive-peak-normalized HRF shape weights with one row per basis function and one column per voxel.
amplitude_scale	The signed scale removed from each raw pooled-fit coefficient column.
degenerate	Logical, one per voxel: the estimated shape has no positive peak after orientation, or one smaller than 5% of its largest absolute deflection (typically a voxel without signal). Such shapes are scaled by their largest absolute value instead, and a warning reports how many there are; estimation does not fail.
basis	The HRF basis object used.
conditions	Observed event labels. Labels are metadata: all events are pooled into one shape per voxel.
condition_pooling	The literal string "all-events".

Examples

```
set.seed(1)
Y <- matrix(rnorm(100), 50, 2)
events <- data.frame(onset = c(5, 25), duration = 1,
                     condition = "A")
basis <- fmrihrf::HRF_SPMG1
sframe <- fmrihrf::sampling_frame(blocklens = nrow(Y), TR = 1)
times <- fmrihrf::samples(sframe, global = TRUE)
rset <- fmrihrf::regressor_set(onsets = events$onset,
                              fac = factor(rep("all events", nrow(events))),
                              hrf = basis, duration = events$duration,
                              span = 30)
X <- fmrihrf::evaluate(rset, grid = times, precision = 0.1, method = "conv")
coef <- matrix(rnorm(ncol(X) * ncol(Y)), ncol(X), ncol(Y))
Y <- X %*% coef + Y * 0.1
est <- estimate_voxel_hrf(Y, events, basis, sframe = sframe)
str(est)
```

fit_oasis_grid

Fit OASIS with HRF Grid Search

Description

Selects an LWU HRF by joint-model profile fit, then estimates OASIS betas.

Usage

```
fit_oasis_grid(Y, onsets, sframe, hrf_grid, ridge_x = 0.01, ridge_b = 0.01)
```

Arguments

Y	Data matrix (time x voxels)
onsets	Event onset times
sframe	Sampling frame
hrf_grid	List of HRF models to test
ridge_x	Ridge parameter for design matrix
ridge_b	Ridge parameter for aggregator

Details

Each candidate is scored by pooled R-squared from unpenalized joint least squares with a voxel-specific intercept and all trial columns. The denominator sums squared deviations from each voxel's own mean. Ridge parameters affect only the final LSS estimates, not HRF selection. Simulation, scoring, and fitting use the same event construction at 0.1-second precision. This is an in-sample selection criterion; overlapping events can leave HRF parameters weakly identifiable.

Value

List with best HRF index, parameters, and beta estimates

Examples

```
onsets <- generate_rapid_design(n_events = 4, total_time = 60, seed = 1)
sim <- generate_lwu_data(onsets, total_time = 60, n_voxels = 2, seed = 1)
grid <- create_lwu_grid(n_tau = 2, n_sigma = 2, n_rho = 2)
fit <- fit_oasis_grid(sim$Y, sim$onsets, sim$sframe, grid)
fit$best_params
```

fmrilss_options

Option constructors for nested interfaces

Description

These helpers create validated option lists for `lss()` and friends.

Value

No value itself. This topic groups the documented constructors `stglmnet_options()`, `oasis_options()`, and `prewhiten_options()`.

Examples

```
stglmnet_options(mode = "fixed", lambda = 0.1)
oasis_options(ridge_x = 0.1, ridge_b = 0.1)
prewhiten_options(method = "ar", p = 1)
```

generate_lwu_data

*Generate Synthetic fMRI Data with LWU HRF***Description**

Creates synthetic fMRI time series using specified LWU HRF parameters

Usage

```
generate_lwu_data(
  onsets,
  tau = 6,
  sigma = 2.5,
  rho = 0.35,
  TR = 1,
  total_time = 300,
  n_voxels = 10,
  amplitudes = NULL,
  noise_sd = 0.2,
  seed = NULL
)
```

Arguments

onsets	Vector of event onset times in seconds
tau	LWU tau parameter (time-to-peak)
sigma	LWU sigma parameter (width)
rho	LWU rho parameter (undershoot amplitude)
TR	Repetition time in seconds
total_time	Total scan time in seconds
n_voxels	Number of voxels to simulate
amplitudes	Event amplitudes (scalar or vector)
noise_sd	Standard deviation of noise
seed	Random seed

Value

List with Y (data matrix), true_hrf, true_betas, and design info

Examples

```
onsets <- generate_rapid_design(n_events = 4, total_time = 60, seed = 1)
sim <- generate_lwu_data(onsets, total_time = 60, n_voxels = 2, seed = 1)
dim(sim$Y)
```

generate_rapid_design *OASIS HRF Recovery Testing Functions*

Description

Functions to test OASIS's ability to recover HRF parameters from rapid event-related designs with overlapping HRFs. Generate Rapid Event-Related Design

Usage

```
generate_rapid_design(  
  n_events = 25,  
  total_time = 300,  
  min_isi = 2,  
  max_isi = 4,  
  seed = NULL  
)
```

Arguments

n_events	Number of events to generate
total_time	Total time in seconds
min_isi	Minimum inter-stimulus interval in seconds
max_isi	Maximum inter-stimulus interval in seconds
seed	Random seed for reproducibility

Details

Creates a rapid event-related design with specified ISI range

Value

Numeric vector of event onset times

Examples

```
generate_rapid_design(n_events = 5, total_time = 40, seed = 1)
```

glmsingle

GLMsingle single-trial response estimation

Description

Estimates single-trial response amplitudes with the GLMsingle procedure (Prince et al., 2022): a library-based HRF per voxel, GLMdenoise noise regressors chosen by cross-validation, and voxel-wise fractional ridge regression chosen by cross-validation over repeated conditions. The implementation reorganises the computation (per-run factorisation, a single pass of data products per stage, compiled cross-validation and sufficient-statistic R^2) so it is much faster than the reference implementation while computing the same estimator.

Usage

```
glmsingle(
  Y,
  design,
  tr,
  stim_dur,
  runs = NULL,
  nuisance = NULL,
  poly_degree = NULL,
  fit_hrf = TRUE,
  hrf_library = NULL,
  denoise = TRUE,
  max_pcs = 10L,
  n_pcs = NULL,
  pc_stop = 1.05,
  pc_voxel_r2 = NULL,
  pc_voxel_mask = NULL,
  noise_pool_r2 = NULL,
  noise_pool_brightness = c(99, 0.1),
  noise_pool_mask = NULL,
  ridge = TRUE,
  ridge_fractions = seq(1, 0.05, by = -0.05),
  ridge_rescale = TRUE,
  ridge_alpha = c("grid", "exact"),
  cv_folds = NULL,
  sessions = NULL,
  percent_signal = TRUE,
  nuisance_in_denoise = c("always", "with_pcs"),
  cv_zero_variance = c("ignore", "glmsingle"),
  singular = c("error", "pinv"),
  pc_cv_all_voxels = FALSE,
  chunk_size = 50000L,
  n_threads = 1L,
  verbose = TRUE
```

)

Arguments

Y	Data: a list of time x voxel matrices (one per run), or a single time x voxel matrix with runs giving the run of each row. Runs may differ in length but must share the voxels.
design	Either a list of time x condition 0/1 matrices, one per run (GLMsingle's format; a 1 marks a trial onset), or a data frame with columns run, onset (seconds from the start of the run; must lie on the TR grid) and condition. Conditions that occur in more than one run drive the cross-validation; with no repeats, noise-PC and ridge selection are switched off with a warning. For matrix Y, event run IDs are matched to runs in data order; for list Y, ascending event run IDs correspond to the list order.
tr	Repetition time in seconds.
stim_dur	Trial duration in seconds (sets the duration of the canonical HRF and of the HRF library).
runs	Run identifier for each row of Y when Y is a single matrix; ignored when Y is a list.
nuisance	Optional nuisance regressors (e.g. motion, physiology): a list with one time x regressor matrix (or NULL) per run. They are included in every model; see nuisance_in_denoise.
poly_degree	Degree of the per-run polynomial drift model: a single integer or one per run. Default (NULL): $\text{round}(\text{run_minutes} / 2)$, as in GLMsingle.
fit_hrf	If TRUE (default), choose the best-fitting HRF from hrf_library for each voxel. If FALSE, use the canonical HRF (glmsingle_hrf()) for every voxel.
hrf_library	Optional time x HRF matrix sampled at the TR (first row at trial onset); columns are scaled to unit peak. Default (NULL): the 20-HRF GLMsingle library, glmsingle_hrf_library() . Ignored when fit_hrf = FALSE.
denoise	If TRUE (default), derive GLMdenoise noise regressors (principal components of a noise pool of voxels) and return the type C model.
max_pcs	Largest number of noise PCs considered (default 10). Capped, with a warning, at the noise pool's rank across runs; an empty or rank-zero pool uses zero PCs and skips the PC-count cross-validation.
n_pcs	Use exactly this many noise PCs instead of choosing the count by cross-validation. Default (NULL): choose by cross-validation.
pc_stop	Stopping factor for the cross-validated PC count (default 1.05): the smallest count whose improvement over zero PCs is at least $1 / \text{pc_stop}$ of the best improvement is chosen. Larger values choose fewer PCs.
pc_voxel_r2	ON-OFF R^2 (percent) above which a voxel contributes to choosing the PC count. Default (NULL): a data-driven threshold from a two-Gaussian fit to the ON-OFF R^2 distribution (also stored in settings\$tail_threshold).
pc_voxel_mask	Optional logical vector (one per voxel); only TRUE voxels may contribute to choosing the PC count.

noise_pool_r2	ON-OFF R^2 (percent) below which a voxel may enter the noise pool. Default (NULL): the same data-driven threshold as pc_voxel_r2. The threshold is only estimated when denoise = TRUE; with fewer than two distinct finite ON-OFF R^2 values it is their common value (or zero if none is finite).
noise_pool_brightness	Two numbers $c(p, f)$: only voxels whose mean signal exceeds f times the p -th percentile of the mean volume may enter the noise pool (excludes out-of-brain voxels). Default $c(99, 0.1)$.
noise_pool_mask	Optional logical vector (one per voxel); only TRUE voxels may enter the noise pool.
ridge	If TRUE (default), fit voxel-wise fractional ridge regression and return the type D model.
ridge_fracs	Ridge fractions in $(0, 1]$ to choose from by cross-validation; a fraction is the length of the ridge solution relative to the OLS solution ($1 =$ no shrinkage). A single value skips cross-validation and uses that fraction everywhere. Default: 1 to 0.05 in steps of 0.05.
ridge_rescale	If TRUE (default), rescale each voxel's ridge betas with the linear fit (slope, offset) that best matches its unregularised betas, undoing the overall shrinkage while keeping the ridge pattern.
ridge_alpha	How fractions are converted to ridge penalties: "grid" (default; fracridge's interpolation on a log-spaced penalty grid, as GLMsingle) or "exact" (solve for the penalty that gives the fraction exactly; experimental).
cv_folds	Cross-validation folds: a list of integer vectors of runs held out together. Default (NULL): leave one run out.
sessions	Session of each run (one entry per run). Betas are z-scored within session before cross-validation, removing session-level gain differences. Default (NULL): one session.
percent_signal	If TRUE (default), express betas as percent signal change of each voxel's mean; if FALSE, in raw data units.
nuisance_in_denoise	How nuisance regressors enter the GLMdenoise and ridge fits: "always" (default, recommended) or "with_pcs", which drops them when zero PCs are used (GLMsingle's behaviour).
cv_zero_variance	Treatment, in cross-validation, of voxels whose reference betas have zero variance within a session (e.g. all-zero voxels): "ignore" (default; they carry no weight) or "glmsingle" (reproduce GLMsingle's Python behaviour).
singular	What to do when a run's trial regressors are linearly dependent (e.g. two trials with the same onset): "error" (default, as GLMsingle) or "pinv" (minimum-norm solution with a warning).
pc_cv_all_voxels	If TRUE, run the PC-count cross-validation for every voxel rather than only those that decide the count. Estimates are identical; only the glmbadness diagnostic is filled for all voxels. Default FALSE (faster).

chunk_size	Number of voxels processed at a time (default 50000). Lower it to reduce peak memory; results do not depend on it.
n_threads	Threads for the per-voxel C++ loops (default 1; 0 = OpenMP default). Results do not depend on the thread count. Matrix products use the BLAS library's own threads; use n_threads > 1 only with a single-threaded BLAS, because the two thread pools compete for cores.
verbose	Print progress messages (default TRUE).

Details

Four models are returned, matching GLMsingle:

typea ON-OFF model: one canonical-HRF regressor for all trials.

typeb Single-trial OLS with the best library HRF per voxel.

typec Type B plus GLMdenoise noise regressors.

typed Type C plus voxel-wise fractional ridge regression.

Arguments are grouped by stage: HRF (`fit_hrf`, `hrf_library`), noise regressors (`denoise`, `max_pcs`, `n_pcs`, `pc_*`, `noise_pool_*`), ridge (`ridge`, `ridge_*`) and cross-validation (`cv_folds`, `sessions`, `cv_zero_variance`). The defaults are GLMsingle's defaults and are a good choice for typical event-related designs; the arguments most often worth setting are nuisance (e.g. motion), `cv_folds/sessions` for multi-session data, and `chunk_size` when memory is tight.

Value

An object of class `glmsingle_fit`: a list with elements `typea`, `typeb`, `typec`, `typed` (each a list using GLMsingle's field names; betas are trials x voxels; `typec/typed` are NULL when `denoise/ridge` is FALSE), `meanvol`, `hrf_library`, `hrf_assumed`, `design` (trial bookkeeping), `settings` (the resolved settings, including data-driven thresholds) and `timing`.

Correspondence with GLMsingle options

`fit_hrf` = `wantlibrary`; `denoise` = `wantglmnoise`; `ridge` = `wantfracridge`; `ridge_fractions` = `fracs`; `ridge_rescale` = `wantautoscale`; `max_pcs` = `n_pcs`; `pc_stop` = `pcstop` (a fixed count, GLMsingle's `pcstop` = -k, is `n_pcs` = k); `pc_voxel_r2` = `pcR2cutoff`; `pc_voxel_mask` = `pcR2cutoffmask`; `noise_pool_r2` = `brainR2`; `noise_pool_brightness` = `brainthresh`; `noise_pool_mask` = `brainexclude`; `cv_folds` = `xvalscheme` (1-based run indices); `sessions` = `sessionindicator`; `nuisance` = `extra_regressors`; `poly_degree` = `maxpolydeg`; `percent_signal` = `wantpercentbold`; `chunk_size` = `chunklen`; `stim_dur` = `stimdur`. Output fields keep GLMsingle's names.

Differences from GLMsingle

Defaults reproduce GLMsingle (Python, commit 1ab54a6) except where GLMsingle is internally inconsistent:

- `nuisance_in_denoise` = "always" keeps nuisance regressors in every GLMdenoise and ridge fit; GLMsingle (Python) drops them when zero noise PCs are used, including from the cross-validation reference. `"with_pcs"` reproduces GLMsingle.

- `cv_zero_variance = "ignore"` gives voxels with zero beta variance no weight in cross-validation; GLMsingl's in-place division helper treats them inconsistently across candidates. "glmsingle" reproduces it.
- The ON-OFF R^2 threshold uses a deterministic two-Gaussian mixture fit with a small variance floor (as in GLMsingl's MATLAB code); GLMsingl's Python fit is unseeded.
- `percent_signal` applies to all model types (GLMsingl always scales types C and D).
- Computation is in double precision (GLMsingl uses single).
- HRF indices are 1-based; betas are trials x voxels.

The FIR diagnostic model, `hrfmodel = "optimize"`, `wantlss`, `bootstrap` modes, figures and file outputs are not implemented.

References

Prince, J. S., Charest, I., Kurzawski, J. W., Pyles, J. A., Tarr, M. J., & Kay, K. N. (2022). Improving the accuracy of single-trial fMRI response estimates using GLMsingl. *eLife*, 11, e77599.

Examples

```
set.seed(1)
tr <- 1; n_time <- 120; n_vox <- 40
design <- lapply(1:3, function(r) {
  D <- matrix(0, n_time, 6)
  onsets <- seq(5, 95, by = 8)
  D[cbind(onsets, rep_len(sample(6), length(onsets)))] <- 1
  D
})
Y <- lapply(design, function(D) {
  X <- apply(D, 2, function(s) stats::filter(s, glmsingle_hrf(3, tr),
                                             sides = 1, circular = FALSE))

  X[is.na(X)] <- 0
  100 + X %*% matrix(rnorm(6 * n_vox, 2), 6) + matrix(rnorm(n_time * n_vox), n_time)
})
fit <- glmsingle(Y, design, tr = tr, stim_dur = 3, max_pcs = 2, verbose = FALSE)
dim(fit$typed$betasmd)
```

glmsingle_design

fmridesign front end for glmsingle()

Description

Fits `glmsingle()` using an `fmridesign::event_model()` to describe the trials. The event model must contain one event term with a single factor whose levels are the conditions (repeated conditions drive cross-validation). Onsets must lie on the TR grid.

Usage

```
glmsingle_design(Y, event_model, baseline_model = NULL, stim_dur = NULL, ...)
```

Arguments

<code>Y</code>	Time x voxel data matrix covering all runs of the sampling frame.
<code>event_model</code>	An <code>event_model</code> from <code>fmriddesign</code> .
<code>baseline_model</code>	Optional <code>baseline_model</code> from <code>fmriddesign</code> .
<code>stim_dur</code>	Trial duration in seconds. Default: the event durations, which must then be a single positive value.
<code>...</code>	Further arguments passed to <code>glmsingle()</code> .

Details

A `baseline_model` contributes its nuisance term (e.g. motion) as nuisance. Its drift and block terms are not used: GLMsingle models drift with its own per-run polynomials (`poly_degree`).

Value

A `glmsingle_fit`; see `glmsingle()`.

Examples

```
set.seed(1)
sf <- fmrihrf::sampling_frame(c(80L, 80L), TR = 1)
events <- data.frame(onset = rep(c(5, 20, 35, 50), 2),
                     run = rep(1:2, each = 4),
                     stimulus = factor(rep(c("A", "B"), 4)))
em <- fmriddesign::event_model(
  onset ~ fmriddesign::hrf(stimulus), data = events, block = ~run,
  sampling_frame = sf, durations = rep(2, nrow(events)))
Y <- matrix(100 + rnorm(160 * 4), 160, 4)
fit <- glmsingle_design(Y, em, denoise = FALSE,
                      ridge = FALSE, verbose = FALSE)
dim(coef(fit, type = "b"))
```

`glmsingle_hrf_library` *GLMsingle canonical HRF and HRF library*

Description

Reproduces GLMsingle's canonical HRF (`getcanonicalhrf`) and its library of 20 HRFs (`getcanonicalhrflibrary`) for a stimulus of duration `stim_dur` sampled at `tr`. Each HRF is peak-normalised to 1, and the first sample is coincident with stimulus onset.

Usage

```
glmsingle_hrf_library(stim_dur, tr)

glmsingle_hrf(stim_dur, tr)
```

Arguments

`stim_dur` Stimulus duration in seconds (rounded to 0.1 s).
`tr` Repetition time in seconds.

Value

`glmsingle_hrf()` returns a numeric vector; `glmsingle_hrf_library()` returns a time x 20 matrix.

References

Prince, J. S., et al. (2022). Improving the accuracy of single-trial fMRI response estimates using GLMsingle. *eLife*, 11, e77599.

Examples

```
lib <- glmsingle_hrf_library(stim_dur = 3, tr = 1)
dim(lib)
```

item_build_design	<i>Build ITEM design metadata</i>
-------------------	-----------------------------------

Description

Build and validate trial-wise design objects used by the ITEM helper layer. The returned object has class `item_bundle` and carries trial-level metadata needed by `item_cv()` and `item_slice_fold()`.

Usage

```
item_build_design(
  X_t,
  T_target = NULL,
  run_id = NULL,
  C_transform = NULL,
  trial_id = NULL,
  trial_hash = NULL,
  meta = list(),
  diagnostics = list(),
  validate = TRUE
)
```

Arguments

`X_t` Numeric trial-wise design matrix with shape `n_time` x `n_trials`.
`T_target` Optional supervised targets with `n_trials` rows. Accepts: numeric matrix/data.frame, numeric vector (regression), or factor/character/logical vector (classification labels).

run_id	Optional run/session identifier of length n_trials. If NULL, all trials are assigned to a single run.
C_transform	Optional transformation matrix used to map X_t to the working design matrix X. Must have n_trials rows when provided.
trial_id	Optional trial identifier vector of length n_trials. Defaults to colnames(X_t) when available, else Trial_1..Trial_n.
trial_hash	Optional hash used by alignment guards. If supplied, item_cv(..., check_hash = TRUE) validates it.
meta	Optional metadata list.
diagnostics	Optional diagnostics list to attach to the bundle.
validate	Logical; when TRUE run strict structure checks.

Value

An object of class item_bundle with fields: Gamma, X_t, C_transform, T_target, U, U_by_run, run_id, trial_id, trial_hash, trial_info, meta, and diagnostics.

Examples

```
bundle <- item_build_design(
  X_t = diag(4),
  T_target = factor(c("A", "B", "A", "B")),
  run_id = c(1, 1, 2, 2)
)
bundle$trial_info
```

item_compute_u	<i>Compute ITEM trial covariance matrix</i>
----------------	---

Description

Compute the trial covariance term as the inverse of a ridge-stabilized normal-equation system, using stable solve paths with fallbacks.

Usage

```
item_compute_u(
  X_t,
  V = NULL,
  v_type = c("cov", "precision"),
  ridge = 0,
  method = c("chol", "svd", "pinv"),
  run_id = NULL,
  output = c("matrix", "by_run"),
  tol = sqrt(.Machine$double.eps)
)
```

Arguments

<code>X_t</code>	Numeric trial-wise design matrix (<code>n_time</code> x <code>n_trials</code>).
<code>V</code>	Optional temporal covariance/precision object. Accepted forms: • NULL (identity), • dense/sparse matrix (<code>n_time</code> x <code>n_time</code>), • run-block list of square matrices whose block sizes sum to <code>n_time</code>.
<code>v_type</code>	Whether <code>V</code> is covariance (" <code>cov</code> ") or precision (" <code>precision</code> ").
<code>ridge</code>	Non-negative ridge term added to the trial-wise system matrix before inversion.
<code>method</code>	Preferred solver path (" <code>chol</code> ", " <code>svd</code> ", or " <code>pinv</code> ").
<code>run_id</code>	Optional trial-level run ids (length <code>n_trials</code>). Required only when <code>output = "by_run"</code> .
<code>output</code>	Return full <code>U</code> (" <code>matrix</code> ") or split run blocks (" <code>by_run</code> ").
<code>tol</code>	Numerical tolerance for rank/solver fallbacks.

Value

Numeric matrix `U` by default, or a named list `U_by_run` when `output = "by_run"`. Returned object includes `item_diagnostics` attribute with rank/condition/solver details.

Examples

```
X_t <- diag(4)
item_compute_u(X_t)
item_compute_u(X_t, run_id = c(1, 1, 2, 2), output = "by_run")
```

item_cv

Crossvalidated ITEM decoding

Description

Run deterministic leave-one-run-out (LOSO) crossvalidation for classification or regression using ITEM decoder weights.

Usage

```
item_cv(
  Gamma,
  T_target = NULL,
  U = NULL,
  run_id = NULL,
  mode = c("classification", "regression"),
  metric = NULL,
  ridge = 0,
```

```

    method = c("chol", "svd", "pinv"),
    class_levels = NULL,
    trial_id = NULL,
    trial_hash = NULL,
    check_hash = FALSE
  )

```

Arguments

Gamma	Trial-wise beta matrix ($n_trials \times n_features$) or an item_bundle object.
T_target	Supervised targets ($n_trials \times p$) or target vector. Ignored when Gamma is an item_bundle.
U	Trial covariance as full matrix ($n_trials \times n_trials$) or run-block list. Ignored when Gamma is an item_bundle.
run_id	Run/session id vector (n_trials). Ignored when Gamma is an item_bundle.
mode	Decoding mode: "classification" or "regression".
metric	Optional metric name. Classification: "accuracy" (default), "balanced_accuracy". Regression: "correlation" (default), "rmse".
ridge	Ridge passed to item_fit().
method	Solver preference for item_fit().
class_levels	Optional fixed class order for classification.
trial_id	Optional trial id vector (used if Gamma is not a bundle).
trial_hash	Optional trial hash (used if Gamma is not a bundle).
check_hash	Logical; validate stored trial hash before CV.

Value

Object of class item_cv_result with per-fold metrics, aggregate metric summary, and trial-level predictions.

Examples

```

Gamma <- matrix(
  c(1, 0,
    0.9, 0.1,
    0.1, 0.9,
    0, 1),
  ncol = 2,
  byrow = TRUE
)
item_cv(
  Gamma = Gamma,
  T_target = factor(c("A", "A", "B", "B")),
  U = diag(4),
  run_id = c(1, 1, 2, 2)
)

```

item_fit

Fit ITEM decoder weights

Description

Fit ITEM weights with a ridge-stabilized generalized least-squares solve.

Usage

```
item_fit(
  Gamma_train,
  T_train,
  U_train,
  ridge = 0,
  method = c("chol", "svd", "pinv"),
  tol = sqrt(.Machine$double.eps)
)
```

Arguments

Gamma_train	Numeric matrix (n_train x n_features).
T_train	Numeric target matrix (n_train x p).
U_train	Trial covariance (n_train x n_train) or run-block list.
ridge	Non-negative ridge term added to the left-hand system.
method	Preferred solver path ("chol", "svd", "pinv").
tol	Numerical tolerance for rank/solver fallbacks.

Value

Numeric weight matrix $\hat{W}$ (n_features x p) with item_diagnostics attribute.

Examples

```
Gamma_train <- matrix(
  c(1, 0,
    0.9, 0.1,
    0.1, 0.9),
  ncol = 2,
  byrow = TRUE
)
T_train <- rbind(c(1, 0), c(1, 0), c(0, 1))
W_hat <- item_fit(Gamma_train, T_train, diag(3))
item_predict(Gamma_train, W_hat)
```

item_from_lsa	<i>Build an ITEM bundle from LS-A estimates</i>
---------------	---

Description

Convenience wrapper that runs `lsa()` to estimate trial-wise amplitudes, computes `U`, and returns an `item_bundle` ready for crossvalidation.

Usage

```
item_from_lsa(
  Y,
  X_t,
  T_target,
  run_id,
  Z = NULL,
  Nuisance = NULL,
  V = NULL,
  v_type = c("cov", "precision"),
  ridge = 0,
  lsa_method = c("r", "cpp"),
  solver = c("chol", "svd", "pinv"),
  u_output = c("matrix", "by_run"),
  C_transform = NULL,
  trial_id = NULL,
  trial_hash = NULL,
  meta = list(),
  validate = TRUE
)
```

Arguments

<code>Y</code>	Numeric data matrix (<code>n_time</code> x <code>n_features</code>).
<code>X_t</code>	Numeric trial-wise design matrix (<code>n_time</code> x <code>n_trials</code>).
<code>T_target</code>	Supervised targets with <code>n_trials</code> rows.
<code>run_id</code>	Trial-level run/session identifier (<code>n_trials</code>).
<code>Z</code>	Optional nuisance matrix passed to <code>lsa()</code> .
<code>Nuisance</code>	Alias for <code>Z</code> in <code>lsa()</code> .
<code>V</code>	Optional covariance/precision for <code>item_compute_u()</code> .
<code>v_type</code>	Whether <code>V</code> is covariance or precision.
<code>ridge</code>	Ridge passed to <code>item_compute_u()</code> .
<code>lsa_method</code>	LS-A backend ("r" or "cpp").
<code>solver</code>	Solver preference for <code>item_compute_u()</code> .
<code>u_output</code>	Return full <code>U</code> matrix or <code>U_by_run</code> blocks.

C_transform	Optional transform matrix used to map X_t to the working design matrix X .
trial_id	Optional trial id vector.
trial_hash	Optional trial hash.
meta	Optional metadata list.
validate	Logical; enforce strict checks before returning.

Value

item_bundle with fields Gamma, X_t , T_{target} , $U/U_{\text{by_run}}$, run_id, meta, and diagnostics.

Examples

```
set.seed(1)
X_t <- diag(4)
Y <- X_t %*% matrix(
  c(1, 0,
    0.8, 0.2,
    0.2, 0.8,
    0, 1),
  ncol = 2,
  byrow = TRUE
) + matrix(rnorm(8, sd = 0.01), 4, 2)
bundle <- item_from_lsa(
  Y = Y,
  X_t = X_t,
  T_target = factor(c("A", "B", "A", "B")),
  run_id = c(1, 1, 2, 2),
  lsa_method = "r"
)
names(bundle)
```

item_predict	<i>Predict targets from ITEM weights</i>
--------------	--

Description

Compute $T_{\text{hat}} = \text{Gamma_test} \%*\% W_{\text{hat}}$.

Usage

```
item_predict(Gamma_test, W_hat)
```

Arguments

Gamma_test	Numeric matrix (n_{test} x n_{features}).
W_hat	Numeric matrix (n_{features} x p).

Value

Numeric matrix of predictions (n_{test} x p).

Examples

```
Gamma_test <- matrix(c(1, 0, 0, 1), ncol = 2, byrow = TRUE)
W_hat <- diag(2)
item_predict(Gamma_test, W_hat)
```

item_slice_fold	<i>Slice an ITEM bundle into train/test fold objects</i>
-----------------	--

Description

Create deterministic leave-one-run-out slices for Gamma, T_target, and trial covariance (U or U_by_run).

Usage

```
item_slice_fold(bundle, test_run, check_hash = FALSE)
```

Arguments

bundle	Object of class item_bundle.
test_run	Run/session id to hold out for testing.
check_hash	Logical; if TRUE, validate stored trial_hash.

Value

A list with train/test slices: Gamma_train, Gamma_test, T_train, T_test, U_train, U_test, train_idx, test_idx, train_runs, test_run.

Examples

```
bundle <- item_build_design(
  X_t = diag(4),
  T_target = factor(c("A", "B", "A", "B")),
  run_id = c(1, 1, 2, 2)
)
bundle$Gamma <- matrix(c(1, 0, 0.8, 0.2, 0.2, 0.8, 0, 1), 4, 2, byrow = TRUE)
bundle$U <- diag(4)
fold <- item_slice_fold(bundle, test_run = 2)
fold$test_idx
```

lsa

*Least Squares All (LSA) Analysis***Description**

Performs a standard multiple regression analysis where all trial regressors are fitted simultaneously. This provides a reference comparison to the Least Squares Separate (LSS) approach.

Usage

```
lsa(Y, X, Z = NULL, Nuisance = NULL, method = c("r", "cpp"))
```

Arguments

Y	A numeric matrix where rows are timepoints and columns are voxels/features. This is the dependent variable data.
X	A numeric matrix where rows are timepoints and columns are trial-specific regressors. Each column represents a single trial or event.
Z	A numeric matrix of nuisance regressors (e.g., motion parameters, drift terms). Defaults to NULL.
Nuisance	An alias for Z, provided for consistency with LSS interface. If both Z and Nuisance are provided, Z takes precedence.
method	Character string specifying the computational method: • "r" - Pure R implementation using <code>lm.fit</code> • "cpp" - Compatibility alias for the numerically identical R implementation; no compiled LSA kernel is currently provided

Details

LSA fits the model $Y = X\beta + Z\gamma + \epsilon$, where all trial regressors in X are estimated simultaneously. This is in contrast to LSS, which fits each trial separately while treating other trials as nuisance regressors.

Value

A numeric matrix of size $T \times V$ containing the beta estimates for each trial regressor (rows) and each voxel (columns). Supplied trial and voxel names must be complete and unique; unnamed inputs receive canonical `Trial_i` and `Voxel_i` names.

Examples

```
n_timepoints <- 100
n_trials <- 10
n_voxels <- 50

X <- matrix(0, n_timepoints, n_trials)
```

```

for(i in 1:n_trials) {
  start <- (i-1) * 8 + 1
  if(start + 5 <= n_timepoints) {
    X[start:(start+5), i] <- 1
  }
}

Y <- matrix(rnorm(n_timepoints * n_voxels), n_timepoints, n_voxels)
true_betas <- matrix(rnorm(n_trials * n_voxels, 0, 0.5), n_trials, n_voxels)
for(i in 1:n_trials) {
  Y <- Y + X[, i] %*% matrix(true_betas[i, ], 1, n_voxels)
}

beta_estimates <- lsa(Y, X)

```

lss

Least Squares Separate (LSS) Analysis

Description

Computes trial-wise beta estimates using the Least Squares Separate approach of Mumford et al. (2012). This method fits a separate GLM for each trial, with the trial of interest plus a single regressor formed by summing all other trials in a one-basis design. A K-basis OASIS model uses one summed other-trial regressor per basis function.

Usage

```

lss(
  Y,
  X,
  Z = NULL,
  Nuisance = NULL,
  method = c("r_optimized", "cpp_optimized", "r_vectorized", "cpp", "naive", "oasis",
    "stglmnet"),
  block_size = 96,
  oasis = list(),
  stglmnet = list(),
  prewhiten = NULL,
  trial_groups = NULL,
  ridge = NULL
)

```

Arguments

Y	A numeric matrix of size $n \times V$ where n is the number of timepoints and V is the number of voxels/variables
---	---

<code>X</code>	A numeric matrix of size $n \times T$ for a one-basis design, with one column per trial. A raw K-basis OASIS design has $n \times (T \times K)$ columns and additionally requires <code>oasis\$K</code> , <code>oasis\$ntrials</code> , and an explicit <code>oasis\$trial_basis_map</code> . When <code>X</code> is an unmodified multi-basis <code>fmridesign::design_matrix()</code> with one event term, its column metadata is used to infer that identity contract and canonicalize rows to trial-major, basis-within-trial order.
<code>Z</code>	A numeric matrix of size $n \times F$ representing common fixed regressors included in every trial-wise model (e.g., intercept, condition effects, block effects). Their coefficients are not returned. If <code>NULL</code> , an intercept-only design is used. Defaults to <code>NULL</code> .
<code>Nuisance</code>	A numeric matrix of size $n \times N$ representing nuisance regressors to be projected out before LSS analysis (e.g., motion parameters, physiological noise). If <code>NULL</code> , no nuisance projection is performed. Defaults to <code>NULL</code> .
<code>method</code>	Character string specifying which implementation to use. Options are: • <code>"r_optimized"</code> - Optimized R implementation (recommended, default) • <code>"cpp_optimized"</code> - Optimized C++ implementation with parallel support • <code>"r_vectorized"</code> - Standard R vectorized implementation • <code>"cpp"</code> - Standard C++ implementation • <code>"naive"</code> - Simple loop-based R implementation (for testing) • <code>"oasis"</code> - OASIS method with HRF support and ridge regularization • <code>"stglmnet"</code> - overlap-aware elastic-net backend using <code>glmnet</code>
<code>block_size</code>	An integer specifying the voxel block size for parallel processing, only applicable when <code>method = "cpp_optimized"</code> . Defaults to 96.
<code>oasis</code>	A list of options for the OASIS method (ridge, SE, design construction, etc.). See Details and oasis_options for the full list. Note: <code>oasis\$whiten</code> is deprecated and ignored with a warning. Use the <code>prewhiten</code> parameter instead for all temporal whitening.
<code>stglmnet</code>	A list of options for the <code>method = "stglmnet"</code> backend. See Details and stglmnet_options for the common fields.
<code>prewhiten</code>	A list of prewhitening options using the fmriAR package, or <code>NULL</code> (no whitening, the default). See Details and prewhiten_options for the full list.
<code>trial_groups</code>	Optional vector (character, factor, or integer) with one condition label per trial, i.e. per column of <code>X</code> . When supplied, each trial-wise model uses one summed "other trials" regressor per condition (the LSS-N variant of Turner et al., 2012), with the trial of interest removed from its own condition's regressor, instead of a single regressor pooling every other trial. This is the model used by Nilearn's and NiBetaSeries' LSS beta series and is more accurate when conditions evoke different responses. Supported by methods <code>"r_optimized"</code> , <code>"cpp_optimized"</code> , <code>"cpp"</code> , and <code>"naive"</code> . Defaults to <code>NULL</code> (classic LSS).
<code>ridge</code>	Optional fractional ridge penalty: one number, or two numbers <code>c(trial, others)</code> for the trial-of-interest and the other-trials coefficients. Each is a fraction of the mean design energy (the <code>ridge_mode = "fractional"</code> convention of OASIS), i.e. the penalty added to the trial diagonal is <code>ridge[1] * mean(c_i'c_i)</code> . Ridge shrinks trial estimates toward zero and can greatly reduce their variance in rapid designs where neighbouring trials overlap; it composes with <code>trial_groups</code>

and prewhiten. Supported by methods "r_optimized", "cpp_optimized", "cpp", and "naive". Defaults to NULL (no ridge).

Details

The LSS approach fits a separate GLM for each trial, where each model includes:

- The trial of interest (from column i of X)
- For a one-basis design, all other trials combined into one summed regressor. A K-basis OASIS model uses a K-column summed block.
- Common fixed regressors (Z matrix), whose coefficients are not returned

Computation. Each LSS estimate is a linear functional of the data, $\hat{\beta}_i = w_i^\top y$. The optimized methods build the $n \times T$ weight matrix W from the (small) trial design and compute all trial betas with one matrix product $W^\top Y$. Because the weights lie in the residual space of the confounds, the data matrix is never residualized or copied, so the cost is a single $O(nTV)$ BLAS call regardless of the number of confounds.

If Nuisance regressors are provided, the rank-revealed combined span $\text{cbind}(Z, \text{Nuisance})$ is projected from both Y and X before fitting. Without a separate Nuisance matrix, Z remains explicitly in every trial-wise model.

When using `method="oasis"`, the following options are available in the `oasis` list (see also [oasis_options](#) for a validated constructor):

- `design_spec`: A list for building trial-wise designs from event onsets using `fmrhrf`. Must contain: `sframe` (sampling frame), `cond` (list with onsets, `hrf`, and optionally `span`), and optionally `others` (list of other conditions to be modeled as nuisances). When provided, X can be NULL and will be constructed automatically.
- `K`: Explicit basis dimension for multi-basis HRF models (e.g., 3 for SPMG3). A raw multi-basis X also requires `ntrials` and `trial_basis_map`. An unmodified multi-basis `fmrdesign::design_matrix()` with one event term is recognized from its metadata; an ordinary raw X is otherwise interpreted as $K=1$.
- `ridge_mode`: Either "fractional" (default) or "absolute". In absolute mode, `ridge_x` and `ridge_b` are used directly as regularization parameters. In fractional mode, they represent fractions of the mean design energy for adaptive regularization.
- `ridge_x`: Ridge parameter for trial-specific regressors (default 0.05). Controls regularization strength for individual trial estimates.
- `ridge_b`: Ridge parameter for the aggregator regressor (default 0.05). Controls regularization strength for the sum of all other trials.
- `return_se`: Logical, whether to return model-based standard errors (default FALSE). This is available only for unpenalized OASIS without estimated prewhitening.
- `return_diag`: Logical, whether to return design diagnostics (default FALSE). When TRUE, includes diagnostic information about the design matrix structure.
- `block_cols`: Integer, voxel block size for memory-efficient processing (default 4096). Larger values use more memory but may be faster for systems with sufficient RAM.
- `ntrials`: Required number of trials when a raw $K > 1$ design is supplied.

- `trial_basis_map`: Required data frame for a raw $K > 1$ design, with one row per X column and fields `column`, `trial`, and `basis`.
- `design_spec$hrf_grid`: Candidate HRFs for grid-based selection within an event-built design. A top-level `oasis$hrf_grid` field is invalid and rejected.

Prewhitening (temporal autocorrelation correction):

Use the top-level `prewhiten` parameter for all temporal whitening. This replaces the old `oasis$whiten = "ar1"` syntax, which is now deprecated and ignored. Do *not* put AR options inside the `oasis` list; they belong in `prewhiten`.

When using `method = "stglmnet"`, the backend accepts an additional nested `stglmnet=` list for lambda selection, overlap-adaptive penalties, and optional pooled trial parameterizations. The common pattern is `stglmnet = stglmnet_options(mode = "cv")` to select lambda by cross-validation, or `stglmnet = stglmnet_options(mode = "fixed", lambda = 0.01)` for a fixed elastic-net fit. The backend reuses `fmrilss` prewhitening and nuisance-projection utilities rather than maintaining a separate whitening path.

The `prewhiten` list accepts the following fields (see also [prewhiten_options](#) for a validated constructor):

- `method`: Character, "ar" (default when the list is non-NULL), "arma", or "none". "ar" fits a pure autoregressive model; "arma" adds a moving-average component (requires $q > 0$).
- `p`: AR order. An integer, or "auto" (default) to select via AIC/BIC up to `p_max`. Use $p = 1$ for a simple AR(1) model (the most common choice for fMRI); higher orders are rarely needed but may help with short TRs or multi-band sequences.
- `q`: Integer MA order for ARMA models (default 0). Only relevant when `method = "arma"`.
- `p_max`: Integer, maximum AR order when `p = "auto"` (default 6).
- `pooling`: How AR coefficients are estimated across voxels. One of:
 - "global" (default) A single set of AR coefficients is estimated from the median autocorrelation across all voxels. Fast and usually adequate.
 - "voxel" Voxel-adaptive noise model. Per-voxel residual autocorrelations are estimated, voxels are grouped into `voxel_bins` bins (default 50) of similar autocorrelation, and an AR model is refitted per bin; each bin gets its own filtered design, as in Nilearn's AR(1) GLM. Supported by methods "r_optimized", "cpp_optimized" and "cpp"; other methods reject it.
 - "run" Fit one AR model per run (requires `runs`). Useful when noise structure differs between runs.
 - "parcel" Fit one AR model per parcel (requires `parcels`); each parcel is fitted with its own filtered design. Supported by methods "r_optimized", "cpp_optimized" and "cpp".
- `runs`: Integer vector of length `nrow(Y)` giving run/block labels. Required for `pooling = "run"` and recommended whenever data span multiple runs so that whitening respects run boundaries.
- `parcels`: Integer vector of length `ncol(Y)` giving parcel labels. Required for `pooling = "parcel"`.
- `exact_first`: Character, "ar1" (default) or "none". When "ar1", the first observation of each segment is scaled by $\sqrt{1 - \phi_1^2}$ for the exact likelihood; "none" drops the first observation instead.

- `compute_residuals`: Logical (default TRUE). When TRUE, OLS residuals (see `residual_model`) are computed before fitting the noise model. Set to FALSE only if Y is already residualized.
- `design`: Optional numeric design matrix whose projection produced those residuals. Supplying it opts in to fmriAR's correction for downward bias in residual autocovariance. When `compute_residuals = TRUE`, it must span the same columns as the full X/Z/Nuisance design, including the intercept that fmriAR adds when none is already represented.
- `acvf_correction`: Optional correction matrix or list of matrices from `fmriAR::acvf_bias_matrix()`, used instead of `design` when reusing a correction across datasets. The two fields are mutually exclusive.
- `voxel_bins`: Positive integer number of autocorrelation bins for pooling = "voxel" (default 50).
- `residual_model`: which model's residuals the noise model is estimated from. "aggregate" (default) uses the confounds plus one summed trial regressor per `trial_groups` level (or per basis function); "full" uses the full trial-wise design, whose many columns bias the autocorrelation downward in rapid designs; "corrected" uses the full design with fmriAR's bias correction (least biased, slower, global/run pooling only). "full" is implied by `design/acvf_correction`. See `vignette("prewhitening")` for when to use each.
- `correction_max_lag`: Positive integer lag budget used when `design` is supplied (default 25). The correction is intended for high-pass-filtered designs; without high-pass filtering, the required lag budget can become impractically large. See `fmriAR::fit_noise()` for details.

Typical prewhiten recipes:

```
# Simple AR(1) – good default for most fMRI data
prewhiten = list(method = "ar", p = 1)

# Auto-select AR order (AIC), global pooling
prewhiten = list(method = "ar", p = "auto")

# Per-run AR(1) for multi-run data
prewhiten = list(method = "ar", p = 1, pooling = "run",
                 runs = blockids)

# Or use the validated constructor:
prewhiten = prewhiten_options(method = "ar", p = 1, pooling = "run",
                             runs = blockids)
```

Prewhitening is applied before the LSS analysis to account for temporal autocorrelation in the fMRI time series. Both Y and all design matrices (X, Z, Nuisance) are filtered through the same whitening operator so that OLS on the whitened system is equivalent to GLS on the original data.

The OASIS method provides a mathematically equivalent but computationally optimized version of standard LSS. It reformulates the per-trial GLM fitting as a single matrix operation, eliminating redundant computations. This is particularly beneficial for designs with many trials or when processing large datasets. When $K > 1$ (multi-basis HRFs), the output will have $K \times \text{ntrials}$ rows, with basis functions for each trial arranged sequentially.

Value

Normally, a numeric matrix of trial-wise beta estimates: $T \times V$ for a one-basis design or $(T \times K) \times V$ for OASIS with K basis functions. With OASIS `return_diag = TRUE` or `return_se = TRUE`, returns `list(beta, diag?, se?)`; `beta` and `se` have the same row-by-voxel shape. One-basis diagnostics contain length- T vectors `d`, `alpha`, and `s`; multi-basis diagnostics contain $K \times K \times T$ arrays `D`, `C`, and `E`. Coefficients for common regressors `Z` are not returned. Multi-basis beta and SE matrices carry the canonical `trial_basis_map` attribute. When estimated prewhitening is active, the actual fitted `fmriAR_plan` is available as `attr(result, "whiten_plan")` (and on `result$beta` for structured returns).

References

Mumford, J. A., Turner, B. O., Ashby, F. G., & Poldrack, R. A. (2012). Deconvolving BOLD activation in event-related designs for multivoxel pattern classification analyses. *NeuroImage*, 59(3), 2636-2643.

Turner, B. O., Mumford, J. A., Poldrack, R. A., & Ashby, F. G. (2012). Spatiotemporal activity estimation for multivoxel pattern analysis with rapid event-related designs. *NeuroImage*, 62(3), 1429-1438.

Examples

```
n_timepoints <- 100
n_trials <- 10
n_voxels <- 50

X <- matrix(0, n_timepoints, n_trials)
for(i in 1:n_trials) {
  start <- (i-1) * 8 + 1
  if(start + 5 <= n_timepoints) {
    X[start:(start+5), i] <- 1
  }
}

Y <- matrix(rnorm(n_timepoints * n_voxels), n_timepoints, n_voxels)
true_betas <- matrix(rnorm(n_trials * n_voxels, 0, 0.5), n_trials, n_voxels)
for(i in 1:n_trials) {
  Y <- Y + X[, i] %*% matrix(true_betas[i, ], 1, n_voxels)
}

beta_estimates <- lss(Y, X)

Z <- cbind(1, scale(1:n_timepoints))
beta_estimates_with_regressors <- lss(Y, X, Z = Z)

Nuisance <- matrix(rnorm(n_timepoints * 6), n_timepoints, 6)
beta_estimates_clean <- lss(Y, X, Z = Z, Nuisance = Nuisance)

beta_oasis <- lss(Y, X, method = "oasis",
  oasis = list(ridge_x = 0.1, ridge_b = 0.1,
    ridge_mode = "fractional"))
```

```

result_with_se <- lss(Y, X, method = "oasis",
                     oasis = list(return_se = TRUE,
                                   ridge_mode = "absolute",
                                   ridge_x = 0, ridge_b = 0))
beta_estimates <- result_with_se$beta
standard_errors <- result_with_se$se

sframe <- fmrihrf::sampling_frame(blocklens = nrow(Y), TR = 1.0)

beta_auto <- lss(Y, X = NULL, method = "oasis",
                 oasis = list(
                   design_spec = list(
                     sframe = sframe,
                     cond = list(
                       onsets = c(10, 30, 50, 70),
                       hrf = fmrihrf::HRF_SPMG1,
                       span = 25
                     ),
                     others = list(
                       list(onsets = c(20, 40, 60, 80))
                     )
                   )
                 ))

beta_multibasis <- lss(Y, X = NULL, method = "oasis",
                      oasis = list(
                        design_spec = list(
                          sframe = sframe,
                          cond = list(
                            onsets = c(10, 30, 50, 70),
                            hrf = fmrihrf::HRF_SPMG3,
                            span = 30
                          )
                        ),
                        K = 3
                      ))

```

lss_beta_cpp

Vectorized LSS Beta Computation Using C++

Description

Fast C++ implementation of least squares separate (LSS) beta estimation using vectorized matrix operations. Computes all trial betas in a single pass without loops.

Usage

```
lss_beta_cpp(C_projected, Y_projected)
```

Arguments

`C_projected` Projected trial regressors (n x T) from `project_confounds_cpp`
`Y_projected` Projected data (n x V) from `project_confounds_cpp`

Details

This vectorized implementation computes all LSS betas simultaneously using matrix algebra. It's significantly faster than per-trial loops and automatically benefits from BLAS multithreading. The algorithm handles numerical edge cases by applying finite lower bounds to degenerate denominators.

For best performance on large datasets, ensure your R installation uses optimized BLAS (like OpenBLAS or Intel MKL).

Value

Beta matrix (T x V) with LSS estimates for each trial and voxel

Examples

```
n <- 40; V <- 3; T <- 4
X_confounds <- cbind(1, seq_len(n))
Y_data <- matrix(rnorm(n * V), n, V)
C_trials <- matrix(rnorm(n * T), n, T)
result <- project_confounds_cpp(X_confounds, Y_data, C_trials)
betas <- lss_beta_cpp(result$Q_dmat_ran, result$residual_data)
```

`lss_cpp_optimized` *A wrapper for the optimized C++ LSS implementation*

Description

A wrapper for the optimized C++ LSS implementation

Usage

```
lss_cpp_optimized(Y, bdes)
```

Arguments

`Y` the voxel by time data matrix
`bdes` the block design list created by `block_design`

Value

a matrix of beta estimates

Examples

```

set.seed(1)
Y <- matrix(rnorm(16), 8, 2)
X_trials <- matrix(0, 8, 2)
X_trials[2:3, 1] <- 1
X_trials[5:6, 2] <- 1
bdes <- list(
  dmat_base = matrix(1, 8, 1),
  dmat_fixed = NULL,
  dmat_ran = X_trials
)
lss_cpp_optimized(Y, bdes)

```

lss_design

*LSS Analysis with fmridesign Objects***Description**

Perform Least Squares Separate (LSS) analysis using `event_model` and `baseline_model` objects from the `fmridesign` package. This provides a streamlined interface for complex designs with multi-condition, parametric modulators, and structured nuisance handling.

Usage

```

lss_design(
  Y,
  event_model,
  baseline_model = NULL,
  method = "oasis",
  oasis = list(),
  prewhiten = NULL,
  blockids = NULL,
  validate = TRUE,
  ...,
  diagnostics = FALSE
)

```

Arguments

- | | |
|----------------|--|
| Y | Numeric matrix of fMRI data (timepoints × voxels). |
| event_model | An <code>event_model</code> object from <code>fmridesign::event_model()</code> . It must contain exactly one <code>trialwise()</code> target term. Any additional condition-level event terms are included as common fixed regressors. |
| baseline_model | Optional <code>baseline_model</code> object from <code>fmridesign::baseline_model()</code> . Defines drift correction, block intercepts, and nuisance regressors. If <code>NULL</code> , basic baseline intercepts are auto-injected: per-run intercepts derived from <code>blockids</code> (or the sampling frame) are used to ensure proper baseline modeling. |

method	LSS method to use. All methods accepted by <code>lss()</code> are supported for a one-basis trialwise event model. Multi-basis event models require <code>method = "oasis"</code> .
oasis	List of OASIS-specific options: ridge regularization (<code>ridge_x</code> , <code>ridge_b</code> , <code>ridge_mode</code>), standard errors (<code>return_se</code>), etc. See oasis_options and the Details section of lss for the full list. Note: <code>design_spec</code> must not be supplied when providing <code>event_model</code> and is rejected at the adapter boundary; <code>oasis\$whiten</code> is deprecated — use <code>prewhiten</code> instead.
prewhiten	Optional prewhitening specification as a list (or NULL for no whitening). Controls temporal autocorrelation correction via the fmriAR package. Key fields: <code>method</code> ("ar", "arma", "none"), <code>p</code> (AR order or "auto"), <code>pooling</code> ("global", "voxel", "run", "parcel"), and <code>runs/parcels</code> . See prewhiten_options and lss for full details and examples.
blockids	Optional complete exact-integer block/run identifiers, one per scan. If NULL, run intercepts are derived from the sampling frame.
validate	Logical. If TRUE (default), performs validation checks on design compatibility, collinearity, and temporal alignment.
...	Additional arguments passed to the underlying LSS method.
diagnostics	Logical. If TRUE, attach a <code>diagnostics</code> attribute with trial/basis and voxel identities, retained/excluded counts, non-finite output locations, and rank/conditioning of the assembled input design. The design diagnostics precede whitening and describe the full design, not each trial-specific LSS model. No trials or voxels are silently dropped; invalid inputs raise errors. This option does not change the return type.

Details

Design Specification:

The `event_model` should typically use `trialwise()` for LSS:

```
emod <- event_model(onset ~ trialwise(basis = "spm1"),
  data = events,
  block = ~run,
  sampling_frame = sframe)
```

Non-trial event terms are supported as common fixed regressors, for example a parametric modulator alongside the unique trialwise target term:

```
emod <- event_model(onset ~ trialwise(basis = "spm1") + hrf(RT),
  data = events,
  block = ~run,
  sampling_frame = sframe)
```

Baseline Model:

If provided, `baseline_model` components are mapped as follows:

- drift and block terms → Z parameter (fixed effects)
- nuisance term → Nuisance parameter (confounds)

Multi-Run Handling:

Both `event_model` and `baseline_model` must use the same `sampling_frame`. Event onsets should be run-relative (resetting to 0 each run) as per `fmridesign` convention; conversion to global time and run boundaries in the design are handled automatically. A joint call still uses one OASIS other-trial aggregate across all runs, so its coefficients are cross-run pooled. Fit each run separately when run-local LSS coefficients are required.

Prewhitening:

Use the `prewhiten` parameter (not the `oasis` list) for temporal autocorrelation correction. For active multi-run whitening, this adapter infers the scan-level run segmentation from the sampling frame. An explicit `prewhiten$runs` vector is allowed only when it encodes those same boundaries. The event model's `blockids` usually has one value per event and is not the required scan-level vector. Residual-autocovariance bias correction remains an explicit opt-in: `lss_design()` does not silently populate `prewhiten$design`. If supplied, it must be the assembled residual-forming design, including trialwise, fixed event, baseline, nuisance, and intercept columns as applicable. This preserves `fmriAR`'s requirement that the correction design be the one that actually produced the residuals. See [lss](#) and [prewhiten_options](#) for full details.

Validation:

When `validate = TRUE`, the function checks:

- Temporal alignment: `nrow(Y)` matches total scans in `sampling_frame`
- Collinearity: emits a warning when the full assembled design has a condition number above 30 and all effective ridge penalties are zero. The condition number is scale-dependent and is not returned.
- Compatibility: `event_model` and `baseline_model` use same `sampling_frame`

Value

Normally a trial-by-voxel beta matrix, or a (trial $\times$ basis)-by-voxel matrix for multi-basis HRFs. When `OASIS return_diag = TRUE` or `return_se = TRUE`, returns `list(beta, diag?, se?)` with the matrix/list shapes documented in [lss\(\)](#). Multi-basis rows are always trial-major with basis varying within trial. Adapter metadata are attached to the returned object; multi-basis beta and SE matrices retain the canonical `trial_basis_map`. With active estimated prewhitening, `attr(result, "whiten_plan")` records the fitted `fmriAR_plan`.

See Also

[lss](#) for the traditional matrix-based interface, `fmridesign::event_model` for event model creation, `fmridesign::baseline_model` for baseline model creation

Examples

```
library(fmridesign)
library(fmrihrf)

sframe <- sampling_frame(blocklens = c(150, 150), TR = 2)

trials <- data.frame(
```

```

onset = c(10, 30, 50, 70, 90, 110,
          10, 30, 50, 70, 90, 110),
run = rep(1:2, each = 6)
)

emod <- event_model(
  onset ~ trialwise(basis = "spmgl"),
  data = trials,
  block = ~run,
  sampling_frame = sframe
)

motion <- list(
  matrix(rnorm(150 * 6), 150, 6),
  matrix(rnorm(150 * 6), 150, 6)
)
bmodel <- baseline_model(
  basis = "bs",
  degree = 5,
  sframe = sframe,
  nuisance_list = motion
)

Y <- matrix(rnorm(300 * 1000), 300, 1000)
beta <- lss_design(Y, emod, bmodel, method = "oasis")

dim(beta)

```

lss_fit_wrappers

Convenience wrappers for modern lss() usage

Description

These functions provide a modern-signature entry point for methods that historically required a block-design (bdes) object.

Value

No value itself. This topic groups `lss_naive_fit()` and `lss_optimized_fit()`.

Examples

```

set.seed(1)
Y <- matrix(rnorm(16), 8, 2)
X <- matrix(0, 8, 2)
X[2:3, 1] <- 1
X[5:6, 2] <- 1
lss_naive_fit(Y, X)

```

lss_naive

*Naive Least Squares Separate (LSS) Analysis***Description**

Performs LSS analysis using the naive approach where each trial model is fit separately. This is the conceptually simplest implementation but less efficient than the optimized [lss](#) function.

Usage

```
lss_naive(Y = NULL, bdes, dset = NULL, trial_groups = NULL, ridge = NULL)
```

Arguments

Y	A numeric matrix where rows are timepoints and columns are voxels/features. If NULL, the function will attempt to extract data from dset.
bdes	A list containing design matrices with components: • dmat_base: Base design matrix (e.g., intercept, drift terms) • dmat_fixed: Fixed effects design matrix (optional) • dmat_ran: Random/trial design matrix for LSS analysis • fixed_ind: Indices for fixed effects (optional)
dset	Optional dataset object. If provided and Y is NULL, data will be extracted using <code>get_data_matrix</code> .
trial_groups	Optional vector with one condition label per trial. When supplied, each trial model uses one summed "other trials" regressor per group (LSS-N) instead of a single pooled regressor. See lss .
ridge	Optional fractional ridge penalty (one or two numbers); see lss .

Details

This function implements the LSS approach by fitting a separate GLM for each trial. Following the method described by Mumford et al. (2012), the model for each trial includes:

- The regressor for the trial of interest.
- A single regressor representing all other trials (the sum of their individual regressors).
- All base regressors (e.g., intercept, drift terms).
- All fixed effects regressors (if any).

While less efficient than the optimized [lss](#) function, this implementation is conceptually clear and serves as a reference for validation.

Value

A numeric matrix with dimensions (n_events x n_voxels) containing the LSS beta estimates for each trial and voxel.

See Also

[lss](#) for the optimized implementation

Examples

```
set.seed(1)
n_timepoints <- 20
n_trials <- 4
n_voxels <- 3

X <- matrix(0, n_timepoints, n_trials)
for (i in seq_len(n_trials)) {
  onset <- 1 + (i - 1) * 4
  X[onset:(onset + 2), i] <- 1
}

Z <- cbind(Intercept = 1, Linear = seq_len(n_timepoints))
Y <- matrix(rnorm(n_timepoints * n_voxels), n_timepoints, n_voxels)

for (i in seq_len(n_trials)) {
  Y <- Y + X[, i] %*% matrix(rnorm(n_voxels, sd = 0.2), 1, n_voxels)
}

bdes <- list(
  dmat_base = Z,
  dmat_ran = X,
  dmat_fixed = NULL,
  fixed_ind = NULL
)

beta_estimates_naive <- lss_naive(Y = Y, bdes = bdes)

beta_estimates_fast <- lss(Y = Y, X = X, Z = Z)
max(abs(beta_estimates_naive - beta_estimates_fast))
```

lss_naive_fit

Naive LSS with modern signature

Description

Equivalent to calling `lss(..., method = "naive")`.

Usage

```
lss_naive_fit(Y, X, Z = NULL, Nuisance = NULL, prewhiten = NULL)
```

Arguments

Y	Numeric matrix (timepoints x voxels).
X	Trial design matrix (timepoints x trials).
Z	Optional experimental regressors.
Nuisance	Optional nuisance regressors to project out.
prewhiten	Optional prewhitening options list (see <code>prewhiten_options()</code>).

Value

A numeric matrix (trials x voxels) of beta estimates.

Examples

```
set.seed(1)
Y <- matrix(rnorm(16), 8, 2)
X <- matrix(0, 8, 2)
X[2:3, 1] <- 1
X[5:6, 2] <- 1
lss_naive_fit(Y, X)
```

lss_optimized	<i>Optimized LSS Analysis (Pure R)</i>
---------------	--

Description

An optimized version of the LSS analysis that avoids creating large intermediate matrices, providing a significant speedup and lower memory usage for the pure R implementation.

Usage

```
lss_optimized(Y = NULL, bdes, dset = NULL, use_cpp = TRUE)
```

Arguments

Y	A numeric matrix where rows are timepoints and columns are voxels/features.
bdes	A list containing the design matrices.
dset	Optional dataset object.
use_cpp	Logical. If TRUE (default), uses the C++ implementation. If FALSE, uses the new optimized R implementation.

Value

A numeric matrix of LSS beta estimates.

Examples

```

set.seed(1)
Y <- matrix(rnorm(16), 8, 2)
X_trials <- matrix(0, 8, 2)
X_trials[2:3, 1] <- 1
X_trials[5:6, 2] <- 1
bdes <- list(
  dmat_base = matrix(1, 8, 1),
  dmat_ran = X_trials,
  dmat_fixed = NULL,
  fixed_ind = NULL
)
lss_optimized(Y, bdes, use_cpp = FALSE)

```

lss_optimized_fit	<i>Optimized LSS with modern signature</i>
-------------------	--

Description

This is a convenience wrapper around `lss()` that selects one of the optimized implementations.

Usage

```

lss_optimized_fit(
  Y,
  X,
  Z = NULL,
  Nuisance = NULL,
  engine = c("cpp", "r"),
  block_size = 96,
  prewhiten = NULL
)

```

Arguments

Y	Numeric matrix (timepoints x voxels).
X	Trial design matrix (timepoints x trials).
Z	Optional experimental regressors.
Nuisance	Optional nuisance regressors to project out.
engine	"cpp" (default) for method="cpp_optimized" or "r" for method="r_optimized".
block_size	Block size used by the C++ optimized path.
prewhiten	Optional prewhitening options list (see <code>prewhiten_options()</code>).

Value

A numeric matrix (trials x voxels) of beta estimates.

Examples

```
set.seed(1)
Y <- matrix(rnorm(16), 8, 2)
X <- matrix(0, 8, 2)
X[2:3, 1] <- 1
X[5:6, 2] <- 1
lss_optimized_fit(Y, X, engine = "r")
```

lss_rank1

Rank-1 GLM: joint voxel-wise HRF and trial amplitude estimation

Description

Fits the rank-1 GLM of Pedregosa et al. (2015): within each voxel one HRF, expressed in an HRF basis, is shared by all trials, and each trial has its own amplitude. With `model = "separate"` (R1-GLMS, the default and the best performer in that paper) every trial is fitted in its own least-squares separate (LSS) model, so the amplitudes are LSS estimates under the voxel's estimated HRF. With `model = "joint"` (R1-GLM) all trials enter one least-squares-all model.

Usage

```
lss_rank1(
  Y,
  events,
  basis,
  sframe,
  nuisance_regs = NULL,
  fixed_regs = NULL,
  model = c("separate", "joint"),
  trial_groups = NULL,
  init = "aggregate",
  ref_hrf = NULL,
  prewhiten = NULL,
  solver = c("als", "lbfgs"),
  max_iter = 100L,
  tol = 1e-07
)
```

Arguments

<code>Y</code>	Numeric matrix of BOLD data (time x voxels).
<code>events</code>	Data frame with onset, duration and condition columns (and run for multi-run sampling frames, with run-relative onsets), as for estimate_voxel_hrf() . Each row is one trial. Conditions are labels only: all trials share each voxel's HRF.
<code>basis</code>	An <code>fmrihrf</code> HRF basis, e.g. <code>fmrihrf::HRF_SPMG3</code> or an FIR basis from <code>fmrihrf::hrf_fir_generator</code>

sframe	An fmrihrf sampling frame with <code>nrow(Y)</code> scans.
nuisance_regs	Optional numeric matrix of nuisance regressors.
fixed_regs	Optional numeric matrix of fixed regressors (run intercepts are added when not already spanned).
model	"separate" (R1-GLMS) or "joint" (R1-GLM).
trial_groups	Optional vector with one condition label per trial (row of events), e.g. <code>events\$condition</code> , for <code>model = "separate"</code> . Each trial-wise model then has one "other trials" regressor per group (LSS-N, as in <code>lss()</code>). Because the HRF is learned mostly through these group regressors, supply <code>trial_groups</code> whenever conditions may differ in mean response, in particular in sign. See Details.
init	"aggregate", "reference", or a $K \times V$ numeric matrix (or length- K vector) of starting HRF coefficients.
ref_hrf	HRF used for the "reference" initialization and to orient the sign of each estimated HRF. Defaults to <code>fmrihrf::HRF_SPMG1</code> .
prewhiten	Optional prewhitening options (see <code>prewhiten_options()</code>). Global and run pooling are supported; the noise model is estimated with one aggregate regressor per basis function.
solver	"als" (default): exact alternating least squares, run in parallel over voxels with OpenMP. "lbfgs": joint quasi-Newton optimization of HRF and amplitudes with L-BFGS-B, as in Pedregosa et al. (2015), on the same objective and starting point (serial; separate model only). See Details.
max_iter	Maximum number of alternating iterations (for "lbfgs", at least 1000 quasi-Newton iterations are allowed).
tol	Relative objective change used as the convergence criterion (for "lbfgs", passed as <code>factr = tol / .Machine\$double.eps</code>).

Details

For trial i with basis-convolved design block X_i ($n \times K$) and the sum of all blocks T_X , the separate model minimizes, per voxel,

$$\sum_i \|y - \beta_i X_i h - r_i(T_X - X_i)h\|^2$$

over the HRF coefficients h , trial amplitudes β and other-trial amplitudes r ; the joint model minimizes $\|y - \sum_i \beta_i X_i h\|^2$. Fixed regressors (run intercepts are added when absent) and nuisance regressors are projected from the design and data first, which gives each trial-wise model its own confound coefficients, as in `lss()`. (The original formulation shares one set of confound coefficients across the trial-wise models.)

The fit uses exact alternating least squares rather than the quasi-Newton solver of the paper: given h , the amplitudes are ordinary LSS (or LSA) estimates; given the amplitudes, h solves a K -dimensional least squares problem. Each step minimizes its block exactly, so the objective never increases. All quantities either step needs are $K \times K$ Gram blocks of the residualized design and the product $X^\top Y$, so after that one product the separate model costs $O(TK^2)$ per voxel and iteration, independent of the number of scans. The joint model costs $O(T^2K^2 + T^3)$ per voxel and iteration and is intended for regions of interest.

The problem is bilinear: h and the amplitudes are determined only up to a common scale and sign. After fitting, each voxel's HRF is oriented to correlate positively with `ref_hrf` and scaled to a unit positive peak, and the amplitudes absorb that scale, so they are in peak-response units for unit-amplitude, zero-duration events, like `lss_with_hrf()`. The returned `hrf` is a `VoxelHRF` object and can be passed to `lss_with_hrf()` to estimate trial amplitudes in new data (for example a held-out run) with the learned HRFs.

The objective is not jointly convex. `init = "aggregate"` starts each voxel from the pooled-shape fit of `estimate_voxel_hrf()` (one common amplitude for all trials); `init = "reference"` starts every voxel from the projection of `ref_hrf` onto the basis, which is more stable for voxels whose mean response is near zero. A $K \times V$ matrix of starting coefficients is also accepted.

Value

A list with

beta Trial amplitudes (trials x voxels), in peak-response units.

other For `model = "separate"`, the amplitude of the pooled other-trials regressor in each trial's model (trials x voxels); NULL for the joint model.

hrf A `VoxelHRF` object with the unit-peak HRF coefficients ($K \times \text{voxels}$), the removed `amplitude_scale`, `basis` and `sframe`.

objective Final residual sum of squares per voxel (summed over the trial-wise models for "separate").

iterations, converged Per-voxel iteration counts and convergence flags.

degenerate Voxels whose estimated HRF has no positive peak of at least 5% of its largest absolute deflection after orientation (scaled by that deflection instead, so their amplitudes are not in peak-response units). Also stored in `hrf`. The policy is shared with `estimate_voxel_hrf()`: such voxels are flagged and reported in one warning rather than failing the fit.

model The fitted model.

When prewhitening is applied the fitted plan is attached as the `whiten_plan` attribute.

References

Pedregosa, F., Eickenberg, M., Ciuciu, P., Thirion, B., & Gramfort, A. (2015). Data-driven HRF estimation for encoding and decoding models. *NeuroImage*, 104, 209-220.

See Also

`estimate_voxel_hrf()`, `lss_with_hrf()`, `lss_sbhm()`

Examples

```
set.seed(1)
sframe <- fmrihrf::sampling_frame(blocklens = 200, TR = 1)
events <- data.frame(onset = seq(10, 180, by = 9), duration = 0,
                     condition = "A")
basis <- fmrihrf::HRF_SPMG3
X <- fmrihss:::voxhrf_trial_basis(events, basis, sframe)$X
h <- c(1, 0.4, -0.2)
```

```

beta <- matrix(rnorm(nrow(events) * 5, 1, 0.3), nrow(events), 5)
Y <- X %*% kronecker(beta, h) + matrix(rnorm(200 * 5, sd = 0.5), 200, 5)
fit <- lss_rank1(Y, events, basis, sframe)
dim(fit$beta)
fit$hrf$coefficients[, 1]

```

lss_sbhm

End-to-End LSS with Shared-Basis HRF Matching (SBHM)

Description

Orchestrates the SBHM pipeline: (1) run-safe trial design and OASIS fit in the shared basis, (2) a voxel shape summary from the requested source, (3) hard library matching or explicit blending, and (4) a separate scalar-coefficient refit using the selected voxel shape.

Usage

```

lss_sbhm(
  Y,
  sbhm,
  design_spec,
  Nuisance = NULL,
  prewhiten = NULL,
  prepass = list(),
  match = list(),
  oasis = list(),
  amplitude = list(),
  return = c("amplitude", "coefficients", "both")
)

```

Arguments

<code>Y</code>	Numeric matrix $T \times V$ of fMRI time series.
<code>sbhm</code>	SBHM object from <code>sbhm_build()</code> .
<code>design_spec</code>	List for design construction: <code>list(sframe=..., cond=list(onsets=..., duration=0, span=...), others=list(...))</code> . The target HRF is replaced by the SBHM basis. Multi-run inputs use run-relative onsets and an explicit <code>cond\$run</code> vector; other conditions use the same event fields and may supply their own HRF.
<code>Nuisance</code>	Optional $T \times P$ nuisance regressors.
<code>prewhiten</code>	Optional prewhitening options (see <code>?lss</code>). Run labels are inferred from <code>design_spec\$sframe</code> when omitted; supplied labels must match those sampling-frame boundaries.
<code>prepass</code>	Optional list forwarded to <code>sbhm_prepass()</code> (e.g., <code>ridge</code> , <code>data_fac</code>).
<code>match</code>	Optional list forwarded to <code>sbhm_match()</code> (e.g., <code>shrink</code> , <code>topK</code> , <code>whiten</code> , <code>orient_ref</code>). Additional fields handled here:

	• <code>alpha_source</code>: one of "prepass" (default), "trial_projection", or "oasis_rank1". "trial_projection" estimates voxel shape from per-trial projection coefficients in the shared basis. • <code>rank1_min</code>: optional minimum rank-1 variance fraction in $[0, 1]$ when <code>alpha_source="oasis_rank1"</code>. Voxels below threshold fall back to prepass. • <code>soft_blend</code> logical (default TRUE): when <code>topK > 1</code>, blend the top-K library coordinates per voxel using softmax weights returned by <code>sbhm_match()</code>. If <code>blend_margin</code> is provided, blending is only applied to voxels with <code>margin < blend_margin</code>; others use the hard top-1 assignment. • <code>blend_margin</code> optional numeric threshold on the matching margin for conditional blending. • <code>whiten_power</code> numeric in $[0, 1]$ for partial singular-value whitening (1=full, 0.5=partial). • <code>min_margin</code> optional minimum matching margin. Voxels below threshold fall back to <code>fallback_ref</code>. • <code>min_beta_norm</code> optional minimum norm of the shape summary used for matching. Voxels below threshold fall back to <code>fallback_ref</code>. • <code>fallback_ref</code> optional r-vector fallback coordinate (default <code>sbhm\$ref\$alpha_ref</code>).
<code>oasis</code>	Optional list forwarded to <code>lss(..., method="oasis")</code> . The basis dimension, trial count, run-specific intercept span, and trial/basis map are supplied from the run-safe SBHM design.
<code>amplitude</code>	List controlling the scalar amplitude stage. Fields: • <code>method</code>: one of "lss1" (default), "global_ls", "oasis_voxel". • <code>ridge</code>: for <code>global_ls</code>, either numeric (absolute) or list(mode, lambda). • <code>ridge_frac</code>: for <code>lss1/oasis_voxel</code>, list(x, b) fractional ridge. • <code>cond_gate</code>: optional auto-fallback rule, e.g., list(metric="rho", thr=0.999, fallback="lss1").
<code>return</code>	One of "amplitude", "coefficients", or "both" (default "amplitude").

Details

Most users should treat the prepass, match, oasis, and amplitude inputs as optional nested *override lists*: you can provide only the fields you want to change. Unspecified nested defaults are preserved.

If you already use `fmridesign`, prefer `lss_sbhm_design()` to avoid manually assembling an OASIS design_spec.

Active prewhitening records the fitted `fmriAR_plan` in the `whiten_plan` attribute on the result and its returned amplitude/coefficient arrays. `diag$prewhitening` distinguishes requested and applied whitening, records the residual model, and reports whether filtering changed Y exactly. An applied identity filter can leave the response unchanged; inspect the recorded plan's order and coefficients before interpreting that case.

Value

A list with components:

- amplitude ntrials×V matrix (when requested)
- coeffs_r rxntrials×V array of trial-wise coefficients (when requested)
- matched_name and matched_idx: named top-scoring library identities
- margin named length-V score differences (top1 - top2 cosine); these are not calibrated confidence measures
- alpha_coords rxV matched coordinates per voxel
- shape_mode and fallback_low_conf: named vectors recording the shape policy actually used
- prepass_fallback: named logical vector identifying voxels whose requested shape source fell back to the aggregate prepass
- trial_basis_map: complete trial/basis output identity
- event_amplitude, event_duration, and event_run: named event-design metadata
- diag list with r, ntrials, and times

Scalar amplitudes are coefficients on the supplied event design using the selected rank-truncated library shape. They are not automatically peak BOLD responses. SBHM standard errors are not calibrated and return_se=TRUE fails explicitly for every amplitude method.

See Also

[lss_sbhm_design\(\)](#), [sbhm_prepass\(\)](#), [sbhm_match\(\)](#)

Examples

```
library(fmrihrf)
set.seed(3)
Tlen <- 180; V <- 4
sframe <- sampling_frame(blocklens = Tlen, TR = 1)
H <- cbind(exp(-seq(0, 30, length.out = Tlen)/5),
            exp(-seq(0, 30, length.out = Tlen)/7))
sbhm <- sbhm_build(library_H = H, r = 2, sframe = sframe, normalize = TRUE)
onsets <- seq(8, 140, by = 12)
design_spec <- list(sframe = sframe, cond = list(onsets = onsets, duration = 0, span = 30))
hrf_B <- sbhm_hrf(sbhm$B, sbhm$tgrid, sbhm$span)
rr <- fmrihrf::regressor(onsets = onsets, hrf = hrf_B, duration = 0, span = 30, summate = FALSE)
Xr <- fmrihrf::evaluate(rr, grid = sbhm$tgrid, precision = 0.1, method = "conv")
alpha_true <- rnorm(ncol(sbhm$B))
Y <- matrix(rnorm(Tlen*V, sd = .6), Tlen, V)
Y[,1] <- Y[,1] + Xr %*% alpha_true
out <- lss_sbhm(Y, sbhm, design_spec)
out2 <- lss_sbhm(Y, sbhm, design_spec,
                 match = list(topK = 2, soft_blend = TRUE),
                 return = "amplitude")
names(out)
```

lss_sbhm_design	<i>SBHM Pipeline with fmridesign Models</i>
-----------------	---

Description

Run the SBHM end-to-end pipeline using fmridesign's `event_model` and optional `baseline_model`, mirroring the convenience of `lss_design()` but producing SBHM shape coefficients and (optionally) scalar event-design trial coefficients in the component named `amplitude`.

Usage

```
lss_sbhm_design(
  Y,
  sbhm,
  event_model,
  baseline_model = NULL,
  prewhiten = NULL,
  prepass = list(),
  match = list(),
  oasis = list(),
  amplitude = list(),
  return = c("amplitude", "coefficients", "both"),
  validate = TRUE,
  ...
)
```

Arguments

<code>Y</code>	Numeric matrix $T \times V$ of fMRI time series (timepoints $\times$ voxels).
<code>sbhm</code>	SBHM object as returned by <code>sbhm_build()</code> .
<code>event_model</code>	An <code>event_model</code> from <code>fmridesign::event_model()</code> defining the trial structure (typically created with <code>trialwise()</code>).
<code>baseline_model</code>	Optional <code>baseline_model</code> from <code>fmridesign::baseline_model()</code> . Its drift, block, and nuisance terms are projected out as confounds.
<code>prewhiten</code>	Optional prewhitening options (see <code>?lss</code>).
<code>prepass</code>	Optional list forwarded to <code>sbhm_prepass()</code> .
<code>match</code>	Optional list forwarded to <code>sbhm_match()</code> .
<code>oasis</code>	Optional SBHM OASIS override list. Supported fields are <code>ridge_mode</code> , <code>ridge_x</code> , <code>ridge_b</code> , and <code>block_cols</code> ; false-valued <code>return_se</code> and <code>return_diag</code> are accepted, while true values fail because SBHM uncertainty/design diagnostics are not exposed. The basis rank, trial count, intercept policy, and trial/basis map are fixed internally from <code>sbhm</code> and the event model.
<code>amplitude</code>	Amplitude options (see <code>?lss_sbhm</code>).
<code>return</code>	One of "amplitude", "coefficients", or "both".

validate	Logical; when TRUE, performs basic checks (sampling frame compatibility, temporal alignment) analogous to <code>lss_design()</code> .
...	Reserved for future use.

Details

This function wraps `lss_sbhm()` by locating the unique trialwise event term, rebuilding it with the SBHM basis, and retaining run identities and heterogeneous durations. Non-trial event terms and `baseline_model` columns enter the fixed nuisance span. Multi-run regressors are constructed within runs so HRF tails cannot cross run boundaries.

Value

Same return contract as `lss_sbhm()`.

Examples

```
library(fmridesign)
sframe <- fmrihrf::sampling_frame(blocklens = 60, TR = 1)
trials <- data.frame(onset = c(5, 20, 35), run = 1)
emod <- event_model(onset ~ trialwise(basis = "spmgl"), data = trials,
  block = ~run, sampling_frame = sframe)
times <- fmrihrf::samples(sframe, global = TRUE)
H <- cbind(exp(-times / 5), exp(-times / 7))
sbhm <- sbhm_build(library_H = H, r = 2, sframe = sframe,
  normalize = TRUE, baseline = NULL)
Y <- matrix(rnorm(60 * 4), 60, 4)
out <- lss_sbhm_design(Y, sbhm, emod)
```

lss_with_hrf

Perform LSS using Voxel-wise HRFs

Description

Computes trial-wise beta estimates using voxel-specific HRFs.

Usage

```
lss_with_hrf(
  Y,
  events,
  hrf_estimates,
  nuisance_regs = NULL,
  engine = "R",
  chunk_size = 5000,
```

```

    verbose = TRUE,
    backing_dir = NULL,
    sframe = NULL,
    fixed_regs = NULL
  )

```

Arguments

<code>Y</code>	Numeric matrix of BOLD data (time x voxels).
<code>events</code>	Data frame with onset, duration and condition columns. For a multi-run sampling frame, a run column is required and onsets are physical times relative to that run.
<code>hrf_estimates</code>	A VoxelHRF object returned by <code>estimate_voxel_hrf</code> .
<code>nuisance_regs</code>	Optional finite numeric matrix of nuisance regressors.
<code>engine</code>	Computational engine: "R" for the pure-R implementation (default), or "C++" to request the compiled backend. The returned metadata records the backend actually used after fallback.
<code>chunk_size</code>	Number of voxels to process per batch (C++ engine only).
<code>verbose</code>	Logical; emit progress messages.
<code>backing_dir</code>	Directory for bigmemory backing files. If NULL, a temporary directory is used (C++ engine only).
<code>sframe</code>	Explicit sampling frame. Defaults to the frame stored in <code>hrf_estimates</code> ; no unit-TR fallback is used.
<code>fixed_regs</code>	Optional finite numeric matrix of fixed/common regressors. An intercept is added when absent.

Value

An object of class [LSSBeta](#) for the C++ request, or a numeric matrix (`n_trials` x `n_vox`) for the R engine. Matrix outputs carry sampling-frame, normalization, coefficient-unit, event-amplitude, event-duration, requested-engine, realized-engine, and chunk-size metadata. With a positive-peak-normalized shape, a zero-duration unit-amplitude event has a peak-response-amplitude coefficient. Otherwise the result is a coefficient on the supplied duration- and amplitude-coded event design. The degenerate metadata repeats the per-voxel flags of `hrf_estimates`: flagged voxels have no positive-peak shape, so their coefficients are not in peak-response units.

Examples

```

set.seed(1)
Y <- matrix(rnorm(100), 50, 2)
events <- data.frame(onset = c(5, 25), duration = 1,
                     condition = "A")
basis <- fmrihrf::HRF_SPMG1
sframe <- fmrihrf::sampling_frame(blocklens = nrow(Y), TR = 1)
times <- fmrihrf::samples(sframe, global = TRUE)
rset <- fmrihrf::regressor_set(onsets = events$onset,
                              fac = factor(rep("all events", nrow(events))),

```

```

                                hrf = basis, duration = events$duration,
                                span = 30)
X <- fmrihrf::evaluate(rset, grid = times, precision = 0.1, method = "conv")
coef <- matrix(rnorm(ncol(X) * ncol(Y)), ncol(X), ncol(Y))
Y <- X %*% coef + Y * 0.1
est <- estimate_voxel_hrf(Y, events, basis, sframe = sframe)
betas <- lss_with_hrf(Y, events, est, verbose = FALSE, engine = "R")
dim(betas)

```

LSSBeta

LSSBeta object

Description

Simple list-based S3 class returned by `lss_with_hrf` containing trial-wise beta estimates. `as.matrix()` materializes C++-backed estimates as a dense matrix while preserving voxel names, sampling-frame, normalization, and unit metadata.

Value

No value itself. This topic documents the object returned by `lss_with_hrf(..., engine = "C++")`.

Stored fields

`betas` is the file-backed trial-by-voxel matrix; `dimnames` preserves trial and voxel identity; `sframe`, `normalization`, `units`, `degenerate`, and `event_amplitude` and `event_duration` define timing and coefficient interpretation; and `engine_requested`, `engine_used`, and `chunk_size` record execution. Use `as.matrix()` for the documented dense extraction path; these metadata are retained as matrix attributes.

Examples

```

Y <- matrix(rnorm(100), 50, 2)
events <- data.frame(onset = c(5, 25), duration = 1, condition = "A")
basis <- fmrihrf::HRF_SPMG1
sframe <- fmrihrf::sampling_frame(blocklens = nrow(Y), TR = 1)
est <- estimate_voxel_hrf(Y, events, basis, sframe = sframe)
fit <- lss_with_hrf(Y, events, est, engine = "C++", verbose = FALSE)
class(fit)

```

mixed_precompute	<i>Precompute Workspace for Optimized Mixed Model</i>
------------------	---

Description

Performs expensive matrix computations that don't depend on the response vector, allowing for efficient reuse across multiple voxels.

Usage

```
mixed_precompute(X, Z, K = NULL)
```

Arguments

X	Fixed effects design matrix ($n \times p$)
Z	Random effects design matrix ($n \times q$)
K	Kinship/covariance matrix for random effects ($q \times q$)

Value

Workspace object for use with `mixed_solve_optimized`

Examples

```
X <- matrix(1, 6, 1)
Z <- diag(6)
ws <- mixed_precompute(X, Z)
names(ws)
```

mixed_solve	<i>Mixed Model Solver</i>
-------------	---------------------------

Description

Solves mixed models with random effects using REML or ML estimation. This function provides a unified interface to mixed model estimation, similar to the `lss`/`lsa` functions in this package.

Usage

```
mixed_solve(
  Y,
  X = NULL,
  Z = NULL,
  K = NULL,
  Nuisance = NULL,
  method = c("REML", "ML"),
```

```

    bounds = c(1e-09, 1e+09),
    SE = FALSE,
    return_Hinv = FALSE
)

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

```

Arguments

Y	Response vector or matrix. If a matrix, each column is treated as a separate response variable.
X	Design matrix for fixed effects. If NULL, defaults to intercept only.
Z	Design matrix for random effects. If NULL, defaults to identity matrix.
K	Kinship matrix for random effects. If NULL, defaults to identity matrix.
Nuisance	An alias for X, provided for consistency with lss/lsa interface. If both X and Nuisance are provided, X takes precedence.
method	Character string specifying the estimation method: • "REML" - Restricted Maximum Likelihood (default) • "ML" - Maximum Likelihood
bounds	Numeric vector of length 2 specifying bounds for variance component optimization. Defaults to c(1e-9, 1e9).
SE	Logical, whether to compute and return standard errors. Defaults to FALSE.
return_Hinv	Logical, whether to return the inverse of the H matrix. Defaults to FALSE.

Details

This function fits the mixed model: $Y = X\beta + Zu + \text{error}$, where $u \sim N(0, VuK)$ and $\text{error} \sim N(0, VeI)$. The variance components Vu and Ve are estimated using REML or ML.

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 matrix (if return_Hinv = TRUE).

Examples

```
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(Y, X, Z, K)
```

mixed_solve_optimized *Optimized Mixed Model Solver*

Description

An optimized implementation of mixed model estimation that precomputes expensive matrix operations and can be reused across multiple voxels for significant performance improvements.

Usage

```
mixed_solve_optimized(
  X,
  Z,
  Y,
  K = NULL,
  workspace = NULL,
  compute_se = FALSE,
  n_threads = 0
)
```

Arguments

X	Fixed effects design matrix ($n \times p$)
Z	Random effects design matrix ($n \times q$)
Y	Response data - can be a vector (single voxel) or matrix ($n \times V$ for multiple voxels)
K	Kinship/covariance matrix for random effects ($q \times q$). Defaults to identity.
workspace	Precomputed workspace (optional, will compute if NULL)
compute_se	Whether to compute standard errors (default: FALSE)
n_threads	Number of OpenMP threads for multi-voxel (0 = auto)

Value

List with estimated parameters and variance components

Examples

```
set.seed(1)
X <- matrix(1, 6, 1)
Z <- diag(6)
Y <- matrix(rnorm(12), 6, 2)
ws <- mixed_precompute(X, Z)
fit <- mixed_solve_optimized(X, Z, Y, workspace = ws)
names(fit)
```

oasis_options	<i>Construct OASIS options</i>
---------------	--------------------------------

Description

Convenience constructor for the oasis= list accepted by `lss(method="oasis")`. Unknown fields are rejected so misspelled scientific controls cannot be silently ignored.

Usage

```
oasis_options(
  design_spec = NULL,
  K = NULL,
  ntrials = NULL,
  trial_basis_map = NULL,
  ridge_mode = c("fractional", "absolute"),
  ridge_x = 0.05,
  ridge_b = 0.05,
  block_cols = 4096L,
  return_se = FALSE,
  return_diag = FALSE,
  add_intercept = TRUE,
  hrf_mode = NULL,
  ...
)
```

Arguments

<code>design_spec</code>	Optional design spec list used to build X via <code>fmrihrf</code> .
<code>K</code>	Optional basis dimension override.
<code>ntrials</code>	Number of trials for a raw multi-basis design.
<code>trial_basis_map</code>	For a raw multi-basis design, a data frame with one row per X column and fields column, trial, and basis.

ridge_mode	"fractional" (default) or "absolute".
ridge_x, ridge_b	Non-negative ridge penalties (defaults 0.05 each).
block_cols	Voxel block size for blocked products.
return_se	Logical; return standard errors.
return_diag	Logical; return diagnostics.
add_intercept	Logical; add intercept when Z is NULL.
hrf_mode	Optional mode (e.g. "voxhrf"); advanced use.
...	Certified advanced options: infer_K_from_X, lambda_shape, mu_rough, ref_hrf, shrink_global, orient_ref, and the deprecated whiten compatibility field. Unknown names are rejected.

Value

A list with class "fmrilss_oasis_options".

Examples

```
oasis_options(ridge_mode = "fractional", ridge_x = 0.1, ridge_b = 0.1)
```

plot_hrf_comparison	<i>Plot HRF Recovery Comparison</i>
---------------------	-------------------------------------

Description

Creates visualization comparing true vs recovered HRFs

Usage

```
plot_hrf_comparison(results, save_path = NULL)
```

Arguments

results	Output from compare_hrf_recovery
save_path	Optional path to save plot

Value

A ggplot2 plot object. When save_path is supplied, the same plot is also written to disk.

Examples

```
onsets <- generate_rapid_design(n_events = 4, total_time = 60, seed = 1)
sim <- generate_lwu_data(onsets, total_time = 60, n_voxels = 2, seed = 1)
grid <- create_lwu_grid(n_tau = 2, n_sigma = 2, n_rho = 2)
res <- compare_hrf_recovery(sim, hrf_grid = grid)
plot_hrf_comparison(res)
```

prewhiten_options	<i>Construct prewhitening options</i>
-------------------	---------------------------------------

Description

Convenience constructor for the prewhiten= list accepted by lss().

Usage

```
prewhiten_options(
  method = c("none", "ar", "arma"),
  p = "auto",
  q = 0L,
  p_max = 6L,
  pooling = c("global", "voxel", "run", "parcel"),
  runs = NULL,
  parcels = NULL,
  exact_first = c("ar1", "none"),
  compute_residuals = TRUE,
  design = NULL,
  acvf_correction = NULL,
  correction_max_lag = 25L,
  voxel_bins = 50L,
  residual_model = NULL
)
```

Arguments

method	"none", "ar", or "arma".
p	AR order or "auto".
q	MA order for ARMA.
p_max	Maximum AR order when p="auto".
pooling	"global", "voxel", "run", or "parcel".
runs	Optional complete exact-integer run identifiers. Required for pooling = "run"; execution requires one value per timepoint.

parcels	Optional complete exact-integer parcel identifiers. Required for pooling = "parcel"; execution requires one value per voxel.
exact_first	"ar1" or "none".
compute_residuals	Logical.
design	Optional numeric design matrix whose projection produced the residuals used to estimate the noise model. Supplying it opts in to fmriAR's residual-autocovariance bias correction. It must represent the same column space as the design used for residualization.
acvf_correction	Optional bias-correction matrix, or list of matrices, produced by <code>fmriAR::acvf_bias_matrix()</code> . This is a cached alternative to design; the two options are mutually exclusive.
correction_max_lag	Positive integer lag budget used when design is supplied. See <code>fmriAR::fit_noise()</code> for the computational and filtering requirements of the correction.
voxel_bins	Positive integer number of autocorrelation bins used by pooling = "voxel" in the weight-matrix LSS methods. Voxels with similar residual autocorrelation share one AR model refitted from their pooled residuals (default 50).
residual_model	Which model's OLS residuals the noise model is estimated from. "aggregate" (the default unless design or acvf_correction is supplied) keeps the confounds plus one summed regressor per trial group or basis function: fast and available for every pooling mode, but trial-to-trial response variability stays in the residuals and inflates the autocorrelation estimate when it is large relative to the noise. "full" uses every trial regressor (the LSA model); with many trials this removes so many degrees of freedom that the autocorrelation is biased strongly downward. "corrected" uses the full model and fmriAR's residual-autocovariance bias correction, built automatically; it is the least biased choice but costs roughly $O(n^2 \times \text{trials})$ and requires method = "ar" with global or run pooling. Supplying design or acvf_correction yourself implies "full". See <code>vignette("prewhitening")</code> .

Value

A list with class "fmrilss_prewhiten_options".

Examples

```
prewhiten_options(method = "ar", p = 1, pooling = "run",
  runs = rep(1:2, each = 50))
```

Description

Computes the orthogonal projection matrix $Q = I - X(X'X)^{-1}X'$ that projects out the space spanned by confound regressors X . This is useful for advanced users who want to cache and reuse projection matrices.

Usage

```
project_confounds(X)
```

Arguments

X	Confound design matrix ($n \times p$) where n is number of timepoints and p is number of confound regressors
-----	--

Details

This function uses QR decomposition for numerical stability instead of computing the Moore-Penrose pseudoinverse directly. Only the estimated-rank QR basis is used, so redundant confound columns do not over-project. The resulting matrix Q can be applied to data to remove the influence of confound regressors.

Value

Projection matrix Q ($n \times n$) that projects out the column space of X

Examples

```
n <- 100
X_confounds <- cbind(1, 1:n)
Y_raw <- matrix(rnorm(n * 3), n, 3)

Q <- project_confounds(X_confounds)

Y_clean <- Q %*% Y_raw
```

project_confounds_cpp *Project Out Confounds Using C++*

Description

Fast C++ implementation for projecting out confound variables from data and trial design matrices. This uses Cholesky decomposition for numerical stability and avoids creating large projection matrices.

Usage

```
project_confounds_cpp(X_confounds, Y_data, C_trials)
```

Arguments

<code>X_confound</code> s	Confound design matrix (n x k)
<code>Y_data</code>	Data matrix (n x V) where V is number of voxels
<code>C_trials</code>	Trial design matrix (n x T) where T is number of trials

Details

This function computes residuals $Y - X(X'X)^{-1}X'Y$ and $C - X(X'X)^{-1}X'C$ without explicitly forming the projection matrix $Q = I - X(X'X)^{-1}X'$. This approach uses ~100x less memory for large n and is numerically more stable.

Value

List with projected data (`residual_data`) and projected trials (`Q_dmat_ran`)

Examples

```
n <- 100; V <- 50; T <- 10
X_confound <- cbind(1, seq_len(n), matrix(rnorm(n * 3), n, 3))
Y_data <- matrix(rnorm(n*V), n, V)
C_trials <- matrix(rnorm(n*T), n, T)

result <- project_confound_cpp(X_confound, Y_data, C_trials)
```

sbhm_build

Build a Shared-Basis HRF Library (SBHM)

Description

Learn a low-rank shared time basis from a parameterized HRF library using `fmrirhf::hrf_library()`. The library is evaluated on the TR grid, optionally baseline-removed and L2-normalized, and decomposed via SVD into $B = U_r$ (shared basis), singular values S , and library coordinates $A = \text{diag}(S) \%*\% t(V_r)$.

Usage

```
sbhm_build(
  library_spec = NULL,
  library_H = NULL,
  r = 6,
  sframe = NULL,
  tgrid = NULL,
  span = 32,
  normalize = TRUE,
  baseline = c(0, 0.5),
```

```

    shifts = NULL,
    ref = c("mean", "spm1")
)

```

Arguments

library_spec	Either NULL (when library_H is provided) or a list with: • fun: a function compatible with <code>fmrihrf::hrf_library(fun, pgrid, ...)</code> that returns an <code>fmrihrf</code> HRF object when called with parameters. • pgrid: a <code>data.frame</code> of parameter combinations (see examples), or the object returned by <code>create_lwu_grid()</code>. • span: numeric, HRF span in seconds (default span). • precision: numeric, evaluation precision (default 0.1 sec). • method: evaluation method for <code>fmrihrf::evaluate()</code> (default "conv"). • extras: optional list of additional arguments passed to <code>hrf_library</code>.
library_H	Optional precomputed TxK matrix of candidate HRFs, already aligned to the TR grid tgrid (or sframe). Mutually exclusive with library_spec.
r	Positive integer target rank for the shared basis (default 6). Requests above the numerical library rank are clipped with a warning.
sframe	Optional <code>fmrihrf::sampling_frame</code> , used to derive the global time grid when tgrid is not provided.
tgrid	Optional numeric vector of global times (in seconds). If provided, takes precedence over sframe.
span	HRF span in seconds (default 32). Used for reference HRF when needed.
normalize	Logical, L2-normalize library columns (default TRUE).
baseline	Numeric length-2 vector specifying a time window (in seconds) used for baseline removal (column-wise mean subtraction within this window). Set to NULL to skip.
shifts	Optional numeric vector of time shifts (in seconds). When provided, shifted copies of the library are added by linear interpolation on tgrid.
ref	Reference for coefficient-space shrinkage and orientation. One of "mean" (default) or "spm1". If "spm1", the SPMG1 HRF is projected onto the learned basis to form alpha_ref.

Value

A list with components:

- B (Txr): shared orthonormal time basis
- S (length r): singular values
- A (rxK): coordinates of library HRFs in the shared basis
- tgrid: the global time grid used (seconds)
- span: span used for reference HRF
- ref: list with alpha_ref (length r) and name
- meta: rank, normalization, candidate identity, full singular values, numerical rank, and retained total-energy fraction

Examples

```
library(fmrihrf)
param_grid <- expand.grid(shape = c(6, 8, 10), rate = c(0.9, 1.0, 1.1))
gamma_fun <- function(shape, rate) fmrihrf::as_hrf(
  fmrihrf::hrf_gamma, params = list(shape = shape, rate = rate)
)

sframe <- fmrihrf::sampling_frame(blocklens = 200, TR = 1)
sbhm <- sbhm_build(
  library_spec = list(fun = gamma_fun, pgrid = param_grid, span = 32),
  r = 6, sframe = sframe, baseline = c(0, 0.5)
)

hrf_B <- sbhm_hrf(sbhm$B, sbhm$tgrid, sbhm$span)
```

sbhm_hrf

*Wrap a Learned Basis as an HRF (SBHM HRF)***Description**

Convert a shared time basis matrix B ($T \times r$) into an `fmrihrf::HRF` object so it can be used directly in OASIS design construction. The HRF returns the r basis columns evaluated at arbitrary times by piecewise-linear interpolation on the provided time grid.

Usage

```
sbhm_hrf(B, tgrid, span)
```

Arguments

<code>B</code>	A numeric matrix ($T \times r$) with orthonormal columns (shared basis).
<code>tgrid</code>	Numeric vector of length T giving the global times (seconds) corresponding to the rows of B .
<code>span</code>	Numeric HRF span passed to <code>fmrihrf::HRF()</code> metadata.

Value

An `fmrihrf::HRF` object with `nbasis = ncol(B)`.

Examples

```
tgrid <- 0:30
candidates <- cbind(exp(-tgrid / 5), tgrid * exp(-tgrid / 5))
B <- qr.Q(qr(candidates))
hrf_B <- sbhm_hrf(B, tgrid, span = 30)
fmrihrf::nbasis(hrf_B)
```

sbhm_match

*Match Voxels to Library HRFs in Shared Basis (SBHM)***Description**

Given per-voxel aggregate coefficients `beta_bar` in the shared basis `B`, and library coordinates `A`, perform shape-only matching in a whitened, L2-normalized coefficient space (cosine similarity). Optionally apply a simple shrinkage of `beta_bar` towards a reference coordinate before matching, and an orientation fix.

Usage

```
sbhm_match(
  beta_bar,
  S,
  A,
  shrink = list(tau = 0, ref = NULL, snr = NULL),
  topK = 1,
  whiten = TRUE,
  sv_floor_rel = 1e-06,
  whiten_power = 1,
  orient_ref = TRUE
)
```

Arguments

<code>beta_bar</code>	Numeric matrix ($r \times V$): per-voxel coefficients from a prepass GLM in the SBHM basis. Columns correspond to voxels.
<code>S</code>	Numeric vector (length r): singular values of the library SVD.
<code>A</code>	Numeric matrix ($r \times K$): coordinates of library HRFs in SBHM basis.
<code>shrink</code>	List with optional shrinkage options: • <code>tau</code> numeric in $[0, 1]$: global strength (default 0, i.e., no shrinkage). • <code>ref</code> numeric length-r vector (<code>alpha_ref</code>) or <code>NULL</code>. If <code>NULL</code>, uses the mean of <code>A</code> columns. Shrinkage is: $\text{beta_bar} \leftarrow (1 - \text{lambda}) \text{beta_bar} + \text{lambda} \text{ref}$. • <code>snr</code> optional numeric length-V estimates; if provided, per-voxel $\text{lambda}_v = \text{tau} / (\text{snr}_v + \text{tau})$. Otherwise $\text{lambda} = \text{tau}$.
<code>topK</code>	Positive integer no larger than the library size; return top- K scores/weights if greater than one (default 1).
<code>whiten</code>	Logical, divide coefficients by <code>S</code> before normalization (default <code>TRUE</code>).
<code>sv_floor_rel</code>	Relative singular-value floor used when <code>whiten=TRUE</code> (default $1e-6$).
<code>whiten_power</code>	Numeric in $[0, 1]$ controlling whitening strength when <code>whiten=TRUE</code> . Uses division by $S^{\text{whiten_power}}$ (1 = full whitening, 0.5 = partial whitening, 0 = no whitening). Default 1 .
<code>orient_ref</code>	Logical, flip <code>beta_bar</code> columns when their dot with <code>ref</code> is negative before matching (default <code>TRUE</code>).

Value

A list with:

- `idx` length- V integer indices of best-matching library HRF (1.. K)
- `margin` named length- V numeric score difference: `top1 - top2`. It is not a calibrated confidence measure.
- `alpha_hat` $r \times V$ matrix: the selected library coordinates (unwhitened, unnormalized)
- `scores` optional $K \times V$ cosine score matrix (returned when `topK > 1`)
- `weights` optional cosine-softmax top- K weights per voxel (when `topK > 1`)

Non-finite inputs, zero-norm voxel summaries or library candidates, and malformed matching controls fail explicitly. Named basis axes must identify the same complete set and are aligned before matching.

Examples

```
set.seed(42)
r <- 4; K <- 12; V <- 3
A <- matrix(rnorm(r*K), r, K)
S <- seq(1, 0.2, length.out = r)
alpha2 <- A[,2]
beta_bar <- cbind(alpha2 + rnorm(r, sd = 0.1),
                  A[,7] + rnorm(r, sd = 0.1),
                  A[,10] + rnorm(r, sd = 0.1))
m <- sbhm_match(beta_bar, S, A)
m$idx
```

sbhm_prepass

SBHM Prepass: Aggregate Fit in a Shared Basis

Description

Fit the trial-aggregated SBHM basis to every voxel after projecting run-specific intercepts, supplied nuisance columns, and complete modeled other-condition spans. The same run-safe design builder is used by the full SBHM pipeline.

Usage

```
sbhm_prepass(
  Y,
  sbhm,
  design_spec,
  Nuisance = NULL,
  prewhiten = NULL,
  ridge = list(mode = "fractional", lambda = 0.01, alpha_ref = NULL),
  data_fac = NULL
)
```

Arguments

<code>Y</code>	Finite numeric T by V response matrix.
<code>sbhm</code>	Object returned by <code>sbhm_build()</code> .
<code>design_spec</code>	Event specification with <code>sframe</code> and <code>cond</code> ; multi-run inputs use run-relative onsets and an explicit <code>cond\$run</code> vector.
<code>Nuisance</code>	Optional T by P nuisance matrix.
<code>prewhiten</code>	Optional prewhitening options passed to the package prewhitening layer. Run labels are inferred from <code>design_spec\$sframe</code> when omitted; supplied labels must match those sampling-frame boundaries.
<code>ridge</code>	List with mode, nonnegative lambda, and optional rank-length <code>alpha_ref</code> .
<code>data_fac</code>	Optional exact or approximate factorization with scores T by q and loadings q by V. Active prewhitening is unsupported with this shortcut, and the original full Y remains required by <code>lss_sbhm()</code> . If factor axes are named, scores and loadings must provide the same complete unique names and are aligned before multiplication. When Y is named, loadings must provide the same complete unique voxel names and are reordered to Y.

Value

A list containing named `beta_bar` (rank by voxel), the residualized aggregate design `A_agg`, its Gram matrix `G`, and design diagnostics and identity maps. Active whitening is recorded in the `whiten_plan` attribute.

Examples

```
times <- seq(0, 30, by = 0.5)
H <- cbind(stats::dgamma(times, 5, 1), stats::dgamma(times, 7, 1))
basis <- sbhm_build(library_H = H, tgrid = times, span = 30, r = 2)
spec <- list(sframe = fmrihrf::sampling_frame(80L, TR = 1),
            cond = list(onsets = c(5, 20, 35, 50), duration = 0))
set.seed(1)
pre <- sbhm_prepass(matrix(rnorm(80 * 3), 80, 3), basis, spec)
dim(pre$beta_bar)
```

sbhm_project

Project Trial-wise SBHM Coefficients to Scalar Amplitudes

Description

Given trial-wise coefficients in the shared basis ($\text{rxntrials} \times \text{V}$) and the voxel-specific matched library coordinates `alpha_hat` (rxV), compute scalar amplitudes per trial and voxel via least-squares projection: $a = (\text{alpha}' \text{ beta}) / (\text{alpha}' \text{ alpha})$.

Usage

```
sbhm_project(beta_rt, alpha_hat)
```

Arguments

beta_rt	3D array of shape $r \times \text{ntrials} \times V$ containing per-trial coefficients in the SBHM basis (as returned by OASIS with $K=r$, reshaped).
alpha_hat	Numeric matrix $r \times V$ of matched library coordinates per voxel (e.g., <code>sbhm_match()\$alpha_hat</code>). These should be in the same coordinate system as <code>beta_rt</code> (unwhitened, not L2-normalized) for interpretable amplitudes.

Value

Numeric matrix $\text{ntrials} \times V$ of scalar amplitudes. Zero-norm or non-finite coordinates are unidentified and fail explicitly. When basis or voxel names are present, both inputs must provide the same complete unique identities; `alpha_hat` is reordered to `beta_rt`.

Examples

```
set.seed(1)
r <- 2; ntrials <- 3; nvox <- 2
alpha_hat <- matrix(rnorm(r * nvox), r, nvox)
beta_rt <- array(rnorm(r * ntrials * nvox), c(r, ntrials, nvox))
amps <- sbhm_project(beta_rt, alpha_hat)
dim(amps)
```

stglmnet_options

Construct stglmnet backend options

Description

Convenience constructor for the `stglmnet=` list accepted by `lss(method = "stglmnet")`. Advanced fields accepted through `...` are validated; unknown fields are rejected so misspelled scientific controls cannot be silently ignored.

Usage

```
stglmnet_options(
  mode = c("cv", "fixed"),
  alpha = 0.2,
  lambda = NULL,
  overlap_strategy = c("none", "multiplicative", "additive", "hybrid", "threshold"),
  pool_to_mean = FALSE,
  pool_strength = 1,
  pool_mean_penalty = 0,
  whiten = c("inherit", "auto", "never", "always"),
  cv_folds = 5L,
  cv_type.measure = c("auto", "mse", "correlation", "reliability", "composite"),
  cv_select = c("optimal", "1se"),
```

```

    return_fit = FALSE,
    ...
)

```

Arguments

mode	"cv" (default) selects lambda by internal cross-validation, while "fixed" uses the supplied lambda sequence or the smallest fitted lambda when no scalar is provided.
alpha	Elastic-net mixing parameter passed to glmnet.
lambda	Optional lambda sequence (or scalar in fixed mode), in the original response coordinates used by glmnet. When NULL, Gaussian responses are divided by one pooled RMS after nuisance projection before fitting the automatic path. The same scale is reused in every CV fold; coefficients, predictions and MSE are returned in original response units. Automatic lambda values use normalized-response coordinates, recorded in <code>fit\$lambda_scale</code> and <code>fit\$response_scale</code> when <code>return_fit = TRUE</code> . They are not interchangeable with explicitly supplied raw-response lambdas.
overlap_strategy	Trial-overlap penalty mapping. One of "none", "multiplicative", "additive", "hybrid", or "threshold".
pool_to_mean	Logical; reparameterize trial effects into a pooled mean plus orthogonal contrasts.
pool_strength	Penalty multiplier applied to pooled contrasts.
pool_mean_penalty	Penalty applied to the pooled mean coefficient.
whiten	One of "inherit" (default), "auto", "never", or "always". "inherit" uses the top-level <code>prewhiten=</code> argument only.
cv_folds	Number of folds used when mode = "cv".
cv_type.measure	Cross-validation objective.
cv_select	Lambda selection rule in CV mode. "optimal" uses the best-scoring lambda, "1se" applies the one-standard-error rule.
return_fit	Logical; when TRUE, <code>lss(method="stglmnet")</code> returns a list containing beta, fit metadata, and the selected lambda.
...	Certified advanced backend fields such as graph-pooling, overlap-strength, fold, and whitening controls. Unknown names are rejected.

Details

The internal cross-validation scores tune the estimator. They use a shared response scale and full-data nuisance projection; reliability also compares fold estimates with the full-data fit. These scores are not an unbiased held-out assessment of predictive performance.

Value

A list with class `"fmrilss_stglmnet_options"`.

Examples

```
stglmnet_options(mode = "fixed", lambda = 0.05, alpha = 0.5)
```

VoxelHRF

VoxelHRF object

Description

Simple list-based S3 class returned by `estimate_voxel_hrf` containing voxel-wise HRF basis coefficients and related metadata.

Value

No value itself. This topic documents the structure returned by `estimate_voxel_hrf()`.

Stored fields

`coefficients` contains one normalized HRF-shape column per voxel; `amplitude_scale` records the removed positive-peak scale; `degenerate` flags voxels whose shape has no positive peak of at least 5% of its largest absolute deflection (scaled by that deflection instead); `basis` stores the HRF basis; `conditions` records observed labels while `condition_pooling` states that all events estimate one pooled shape; `sframe` preserves physical scan timing; and `normalization_plus_coefficient_units` state the coefficient scale. Voxel names, when supplied on `Y`, identify coefficient columns and amplitude-scale entries.

Examples

```
Y <- matrix(rnorm(100), 50, 2)
events <- data.frame(onset = c(5, 25), duration = 1, condition = "A")
basis <- fmrihrf::HRF_SPMG1
sframe <- fmrihrf::sampling_frame(blocklens = nrow(Y), TR = 1)
est <- estimate_voxel_hrf(Y, events, basis, sframe = sframe)
class(est)
```

Index

benchmark_mixed_solve, 3

calculate_recovery_metrics, 4

coef.glmsingle_fit, 5

compare_hrf_recovery, 5

create_lwu_grid, 6

create_lwu_grid(), 62

estimate_voxel_hrf, 7

estimate_voxel_hrf(), 43, 45

fit_oasis_grid, 8

fmrilss_options, 9

generate_lwu_data, 10

generate_rapid_design, 11

glmsingle, 12

glmsingle(), 16, 17

glmsingle_design, 16

glmsingle_hrf(glmsingle_hrf_library), 17

glmsingle_hrf(), 13

glmsingle_hrf_library, 17

glmsingle_hrf_library(), 13

item_build_design, 18

item_compute_u, 19

item_cv, 20

item_fit, 22

item_from_lsa, 23

item_predict, 24

item_slice_fold, 25

lsa, 26

lss, 27, 36, 37, 39, 40

lss(), 36, 37, 44

lss_beta_cpp, 33

lss_cpp_optimized, 34

lss_design, 35

lss_fit_wrappers, 38

lss_naive, 39

lss_naive_fit, 40

lss_optimized, 41

lss_optimized_fit, 42

lss_rank1, 43

lss_sbhm, 46

lss_sbhm(), 45, 66

lss_sbhm_design, 49

lss_sbhm_design(), 47, 48

lss_with_hrf, 50

lss_with_hrf(), 45

LSSBeta, 51, 52

mixed_precompute, 53

mixed_solve, 53

mixed_solve_cpp(mixed_solve), 53

mixed_solve_optimized, 55

oasis_options, 28, 29, 36, 56

plot_hrf_comparison, 57

prewhiten_options, 28, 30, 36, 37, 58

prewhiten_options(), 44

project_confounds, 59

project_confounds_cpp, 60

sbhm_build, 61

sbhm_build(), 6, 66

sbhm_hrf, 63

sbhm_match, 64

sbhm_match(), 48

sbhm_prepass, 65

sbhm_prepass(), 48

sbhm_project, 66

stglmnet_options, 28, 67

VoxelHRF, 8, 45, 51, 69