Package: fmrilss (via r-universe)

June 4, 2026

Type Package

Title Least Squares Separate (LSS) Analysis for fMRI Data

Version 0.1.0

Description Implements efficient least squares separate (LSS) analysis for functional magnetic resonance imaging (fMRI) data. LSS is used to estimate trial-by-trial activation patterns in event-related fMRI designs. The package provides both R and C++ implementations for computational efficiency.

License GPL-3

Encoding UTF-8

Depends R (>= 3.5.0)

Imports MASS, Rcpp (>= 0.12.0), fmrihrf (>= 0.1.0), fmriAR (>= 0.0.0.9000), bigmemory, ggplot2, glmnet

LinkingTo Rcpp, RcppArmadillo, roptim, bigmemory, BH

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

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/bbuchsbaum/fmrilss/issues

Language en-US

Config/Needs/website albersdown

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

Date/Publication 2026-05-05 14:04:55 UTC

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

RemoteRef HEAD

RemoteSha e8d305562b96353856e2bea2ff1e72fa66d16805

Contents

benchmark_mixed_solve . . . 3
calculate_recovery_metrics . . . 4
compare_hrf_recovery . . . 4
create_lwu_grid . . . 5
estimate_voxel_hrf . . . 6
fit_oasis_grid . . . 7
fmrilss_options . . . 8
generate_lwu_data . . . 8
generate_rapid_design . . . 9
item_build_design . . . 10
item_compute_u . . . 11
item_cv . . . 13
item_fit . . . 14
item_from_lsa . . . 15
item_predict . . . 17
item_slice_fold . . . 17
lsa . . . 18
lss . . . 19
lss_beta_cpp . . . 24
lss_cpp_optimized . . . 25
lss_design . . . 26
lss_fit_wrappers . . . 29
lss_naive . . . 29
lss_naive_fit . . . 31
lss_optimized . . . 32
lss_optimized_fit . . . 33
lss_sbhm . . . 34
lss_sbhm_design . . . 36
lss_with_hrf . . . 38
LSSBeta . . . 39
mixed_precompute . . . 40
mixed_solve . . . 40
mixed_solve_optimized . . . 42
oasis_options . . . 43
plot_hrf_comparison . . . 44
prewhiten_options . . . 45
project_confounds . . . 46
project_confounds_cpp . . . 47
sbhm_build . . . 48
sbhm_hrf . . . 50
sbhm_match . . . 50

sbhm_prepass 52
sbhm_project 54
stglmnet_options 55
VoxelHRF 56

Index **57**

`benchmark_mixed_solve` *Benchmark Mixed Model Implementations*

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

```
## Not run:
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)

## End(Not run)
```

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

Value

Data frame with recovery metrics

Examples

```
## Not run:
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)

## End(Not run)
```

compare_hrf_recovery *Compare HRF Recovery Methods*

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

```
## Not run:
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)

## End(Not run)
```

---

create_lwu_grid *Create LWU HRF Grid for OASIS Search*

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

List of HRF models and their parameters

Examples

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

`estimate_voxel_hrf` *Estimate Voxel-wise HRF Basis Coefficients*

Description

Fits a GLM to estimate HRF basis coefficients for every voxel.

Usage

```
estimate_voxel_hrf(Y, events, basis, nuisance_regs = NULL)
```

Arguments

`Y`	Numeric matrix of BOLD data (time x voxels).
`events`	Data frame with `onset`, `duration` and `condition` columns.
`basis`	HRF object from the `fmrihrf` package.
`nuisance_regs`	Optional numeric matrix of nuisance regressors.

Value

A VoxelHRF object containing at least:

`coefficients`	Matrix of HRF basis coefficients.
`basis`	The HRF basis object used.
`conditions`	Character vector of modeled conditions.

Examples

```
## Not run:
set.seed(1)
Y <- matrix(rnorm(100), 50, 2)
events <- data.frame(onset = c(5, 25), duration = 1,
                     condition = "A")
basis <- fmrihrf::hrf_gamma()
sframe <- fmrihrf::sampling_frame(blocklens = nrow(Y), TR = 1)
times <- fmrihrf::samples(sframe, global = TRUE)
rset <- fmrihrf::regressor_set(onsets = events$onset,
                               fac = factor(1:nrow(events)),
                               hrf = basis, duration = events$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)
str(est)

## End(Not run)
```

`fit_oasis_grid` *Fit OASIS with HRF Grid Search*

Description

Fits OASIS models with different HRF parameters and selects best

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

Value

List with best HRF index, parameters, and beta estimates

Examples

```
## Not run:
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

## End(Not run)
```

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

```
## Not run:
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)

## End(Not run)
```

---

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)
```

---

item_build_design *Build ITEM design metadata*

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 *Compute ITEM trial covariance matrix*

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

X_t	Numeric trial-wise design matrix (n_time x n_trials).
V	Optional temporal covariance/precision object. Accepted forms: • NULL (identity), • dense/sparse matrix (n_time x n_time), • run-block list of square matrices whose block sizes sum to n_time.
v_type	Whether V is covariance ("cov") or precision ("precision").
ridge	Non-negative ridge term added to the trial-wise system matrix before inversion.
method	Preferred solver path ("chol", "svd", or "pinv").
run_id	Optional trial-level run ids (length n_trials). Required only when output = "by_run".
output	Return full U ("matrix") or split run blocks ("by_run").
tol	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 x n_features) or an item_bundle object.
T_target	Supervised targets (n_trials x p) or target vector. Ignored when Gamma is an item_bundle.
U	Trial covariance as full matrix (n_trials x 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 `W_hat` (`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 *Build an ITEM bundle from LS-A estimates*

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

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

Description

Compute `T_hat = Gamma_test %*% W_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 *Slice an ITEM bundle into train/test fold objects*

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 lm.fit • "cpp" - C++ implementation for better performance

Details

LSA fits the model: Y = X*beta* + *Z*gamma + error, 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 × V containing the beta estimates for each trial regressor (rows) and each voxel (columns).

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 and all other trials as separate regressors.

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
)
```

Arguments

Y	A numeric matrix of size n × V where n is the number of timepoints and V is the number of voxels/variables
X	A numeric matrix of size n × T where T is the number of trials. Each column represents the design for one trial
Z	A numeric matrix of size n × F representing experimental regressors to include in all trial-wise models. These are regressors we want to model and get beta estimates for, but not trial-wise (e.g., intercept, condition effects, block effects). If NULL, an intercept-only design is used. Defaults to NULL
Nuisance	A numeric matrix of size n × N representing nuisance regressors to be projected out before LSS analysis (e.g., motion parameters, physiological noise). If NULL, no nuisance projection is performed. Defaults to NULL
method	Character string specifying which implementation to use. Options are: • "r_optimized" - Optimized R implementation (recommended, default) • "cpp_optimized" - Optimized C++ implementation with parallel support • "r_vectorized" - Standard R vectorized implementation • "cpp" - Standard C++ implementation • "naive" - Simple loop-based R implementation (for testing) • "oasis" - OASIS method with HRF support and ridge regularization • "stglmnet" - overlap-aware elastic-net backend using `glmnet`
block_size	An integer specifying the voxel block size for parallel processing, only applicable when `method = "cpp_optimized"`. Defaults to 96.
oasis	A list of options for the OASIS method (ridge, SE, design construction, etc.). See Details and `oasis_options` for the full list. **Note:** `oasis$whiten` is deprecated and ignored. Use the `prewhiten` parameter instead for all temporal whitening.
stglmnet	A list of options for the `method = "stglmnet"` backend. See Details and `stglmnet_options` for the common fields.
prewhiten	A list of prewhitening options using the **fmriAR** package, or `NULL` (no whitening, the default). See Details and `prewhiten_options` for the full list.

Details

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

- The trial of interest (from column i of X)
- All other trials combined (sum of all other columns of X)
- Experimental regressors (Z matrix) - these are modeled to get beta estimates but not trial-wise

If Nuisance regressors are provided, they are first projected out from both Y and X using standard linear regression residualization.

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 fmrihrf. 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). If not provided, it's auto-detected from X dimensions or defaults to 1 for single-basis HRFs.
- 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 standard errors (default FALSE). When TRUE, returns a list with beta (trial estimates) and se (standard errors) components.
- 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: Explicit number of trials (used when K > 1 to determine output dimensions). If not provided, calculated as ncol(X) / K.
- hrf_grid: Vector of HRF indices for grid-based HRF selection (advanced use). Allows testing multiple HRF shapes simultaneously.

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" Fit a separate AR model per voxel. Most accurate but slow; consider "parcel" instead.

  "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). Good compromise between "global" and "voxel".

- 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 from the full design are computed before fitting the noise model. Set to FALSE only if Y is already residualized.

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)

# Parcel-based AR with atlas labels
prewhiten = list(method = "ar", p = 1, pooling = "parcel",
                 parcels = atlas_labels)

# 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*ntrials rows, with basis functions for each trial arranged sequentially.

Value

A numeric matrix of size T × V containing the trial-wise beta estimates. Note: Currently only returns estimates for the trial regressors (X). Beta estimates for the experimental regressors (Z) are computed but not returned.

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.

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)

## Not run:
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))
beta_estimates <- result_with_se$beta
```

```
standard_errors <- result_with_se$se

  sframe <- sampling_frame(blocklens = 200, 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, 90, 110, 130, 150),
                         hrf = HRF_SPMG1,
                         span = 25
                       ),
                       others = list(
                         list(onsets = c(20, 40, 60, 80, 100, 120, 140))
                       )
                     )
                   ))

  beta_multibasis <- lss(Y, X = NULL, method = "oasis",
                         oasis = list(
                           design_spec = list(
                             sframe = sframe,
                             cond = list(
                               onsets = c(10, 30, 50, 70, 90),
                               hrf = HRF_SPMG3,
                               span = 30
                             )
                           ),
                           K = 3
                         ))

## End(Not run)
```

`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 setting problematic denominators to NaN.

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

```
## Not run:
result <- project_confounds_cpp(X_confounds, Y_data, C_trials)
betas <- lss_beta_cpp(result$Q_dmat_ran, result$residual_data)

## End(Not run)
```

---

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,
  ...
)
```

Arguments

`Y`	Numeric matrix of fMRI data (timepoints × voxels).
`event_model`	An event_model object from `fmridesign::event_model()`. This defines the trial-wise or condition-wise task design. For LSS, typically created with `trialwise()` to generate one regressor per trial.
`baseline_model`	Optional baseline_model object from `fmridesign::baseline_model()`. Defines drift correction, block intercepts, and nuisance regressors. If NULL, basic baseline intercepts are auto-injected: per-run intercepts derived from `blockids` (or the sampling frame) are used to ensure proper baseline modeling.

method	LSS method to use. Currently only "oasis" is supported for event_model integration.
oasis	List of OASIS-specific options: ridge regularization (ridge_x, ridge_b, ridge_mode), standard errors (return_se), etc. See oasis_options and the Details section of lss for the full list. Note: design_spec is not used when providing event_model, and oasis$whiten is deprecated — use prewhiten instead.
prewhiten	Optional prewhitening specification as a list (or NULL for no whitening). Controls temporal autocorrelation correction via the **fmriAR** package. Key fields: method ("ar", "arma", "none"), p (AR order or "auto"), pooling ("global", "voxel", "run", "parcel"), and runs/parcels. See prewhiten_options and lss for full details and examples.
blockids	Optional block/run identifiers for event_model. If NULL, extracted from event_model$blockids.
validate	Logical. If TRUE (default), performs validation checks on design compatibility, collinearity, and temporal alignment.
...	Additional arguments passed to the underlying LSS method.

Details

Design Specification:

The event_model should typically use trialwise() for LSS:

```
emod <- event_model(onset ~ trialwise(basis = "spmg1"),
                    data = events,
                    block = ~run,
                    sampling_frame = sframe)
```

For factorial designs (e.g., estimating condition-level betas separately):

```
emod <- event_model(onset ~ hrf(condition),
                    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. Run structure is automatically respected. Event onsets should be run-relative (resetting to 0 each run) as per fmridesign convention - conversion to global time is handled automatically.

Prewhitening:

Use the prewhiten parameter (not the oasis list) for temporal autocorrelation correction. For multi-run data, pass prewhiten = list(method = "ar", p = 1, pooling = "run", runs = blockids) so that whitening respects run boundaries. 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: Design matrix condition number < 30 (suppressed when ridge is already configured via `oasis$ridge_x` or `oasis$ridge_b`)
- Compatibility: event_model and baseline_model use same sampling_frame

Value

Matrix of trial-wise beta estimates (trials × voxels), or (trials × basis_functions) × voxels for multi-basis HRFs.

See Also

`lss` for the traditional matrix-based interface, `fmridesign::event_model` for event model creation, `fmridesign::baseline_model` for baseline model creation

Examples

```
## Not run:
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 = "spmg1"),
  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)

## End(Not run)
```

`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)
```

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 get_data_matrix.

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 `prewhiten_options()`).

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` *Optimized LSS Analysis (Pure R)*

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` *Optimized LSS with modern signature*

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 `prewhiten_options()`).

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

Description

Orchestrates the SBHM pipeline: (1) prepass aggregate fit in the learned shared basis, (2) cosine matching to a library of HRFs represented in the same basis, (3) Least Squares Separate (OASIS) with the SBHM basis to obtain trial-wise r-dimensional coefficients, and (4) projection of those coefficients onto the matched coordinates to produce scalar amplitudes.

Usage

```
lss_sbhm(
  Y,
  sbhm,
  design_spec,
  Nuisance = NULL,
  prewhiten = NULL,
  prepass = list(),
 match = list(shrink = list(tau = 0, ref = NULL, snr = NULL), topK = 3, soft_blend =
  TRUE, blend_margin = 0.08, whiten = FALSE, sv_floor_rel = 0.05, whiten_power = 0.5,
  min_margin = NULL, min_beta_norm = NULL, fallback_ref = NULL, orient_ref = TRUE,
    alpha_source = "prepass", rank1_min = 0),
  oasis = list(),
 amplitude = list(method = "lss1", ridge = list(mode = "fractional", lambda = 0.02),
  ridge_frac = list(x = 0.02, b = 0.02), cond_gate = NULL, adaptive = list(enable =
    FALSE, base = 0.02, k0 = 1000, max = 0.08), return_se = FALSE),
  return = c("amplitude", "coefficients", "both")
)
```

Arguments

`Y`	Numeric matrix T×V of fMRI time series.
`sbhm`	SBHM object from `sbhm_build()`.
`design_spec`	List for design construction (same as `oasis$design_spec`): `list(sframe=..., cond=list(onsets=..., duration=0, span=...), others=list(...))`. The HRF in `cond$hrf` is ignored and replaced with the SBHM basis HRF.
`Nuisance`	Optional T×P nuisance regressors.
`prewhiten`	Optional prewhitening options (see `?lss`).
`prepass`	Optional list forwarded to `sbhm_prepass()` (e.g., ridge, data_fac).
`match`	Optional list forwarded to `sbhm_match()` (e.g., shrink, topK, whiten, orient_ref). Additional fields handled here: • `alpha_source`: one of `"prepass"` (default), `"trial_projection"`, or `"oasis_rank1"`. `"trial_projection"` estimates voxel shape from per-trial projection coefficients in the shared basis.

- rank1_min: optional minimum rank-1 variance fraction in [0,1] when alpha_source="oasis_rank1". Voxels below threshold fall back to prepass.
- soft_blend logical (default TRUE): when topK > 1, blend the top-K library coordinates per voxel using softmax weights returned by sbhm_match(). If blend_margin is provided, blending is only applied to voxels with margin < blend_margin; others use the hard top-1 assignment.
- blend_margin optional numeric threshold on the matching margin for conditional blending.
- whiten_power numeric in [0,1] for partial singular-value whitening (1=full, 0.5=partial).
- min_margin optional minimum matching margin. Voxels below threshold fall back to fallback_ref.
- min_beta_norm optional minimum norm of the shape summary used for matching. Voxels below threshold fall back to fallback_ref.
- fallback_ref optional r-vector fallback coordinate (default sbhmrefalpha_ref).

oasis — Optional list forwarded to lss(..., method="oasis"). K is set to ncol(sbhm$B) if not provided, and design_spec is injected automatically.

amplitude — List controlling the scalar amplitude stage. Fields:

- method: one of "lss1" (default), "global_ls", "oasis_voxel".
- ridge: for global_ls, either numeric (absolute) or list(mode, lambda).
- ridge_frac: for lss1/oasis_voxel, list(x, b) fractional ridge.
- cond_gate: optional auto-fallback rule, e.g., list(metric="rho", thr=0.999, fallback="lss1").

return — One of "amplitude", "coefficients", or "both" (default "amplitude").

Details

Most users should treat the prepass, match, oasis, and amplitude inputs as optional *override lists*: you can provide only the fields you want to change, and rely on defaults for everything else.

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

Value

A list with components:

- amplitude ntrials×V matrix (when requested)
- coeffs_r r×ntrials×V array of trial-wise coefficients (when requested)
- matched_idx length-V integer indices into the library
- margin length-V confidence margins (top1 - top2 cosine)
- alpha_coords r×V matched coordinates per voxel
- diag list with r, ntrials, and times

See Also

lss_sbhm_design(), sbhm_prepass(), sbhm_match()

Examples

```
## Not run:
  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 = 4, 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 = 3, soft_blend = TRUE),
                   return = "amplitude")
  names(out)

## End(Not run)
```

`lss_sbhm_design` *SBHM Pipeline with fmridesign Models*

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 coefficients and (optionally) scalar amplitudes.

Usage

```
lss_sbhm_design(
  Y,
  sbhm,
  event_model,
  baseline_model = NULL,
  prewhiten = NULL,
  prepass = list(),
 match = list(shrink = list(tau = 0, ref = NULL, snr = NULL), topK = 3, soft_blend =
  TRUE, blend_margin = 0.08, whiten = FALSE, sv_floor_rel = 0.05, whiten_power = 0.5,
  min_margin = NULL, min_beta_norm = NULL, fallback_ref = NULL, orient_ref = TRUE,
    alpha_source = "prepass", rank1_min = 0),
```

```
  oasis = list(),
 amplitude = list(method = "lss1", ridge = list(mode = "fractional", lambda = 0.02),
  ridge_frac = list(x = 0.02, b = 0.02), cond_gate = NULL, adaptive = list(enable =
    FALSE, base = 0.02, k0 = 1000, max = 0.08), return_se = FALSE),
 return = c("amplitude", "coefficients", "both"),
 validate = TRUE,
 ...
)
```

Arguments

Y	Numeric matrix T×V of fMRI time series (timepoints × voxels).
sbhm	SBHM object as returned by sbhm_build().
event_model	An event_model from fmridesign::event_model() defining the trial structure (typically created with trialwise()).
baseline_model	Optional baseline_model from fmridesign::baseline_model(). Its drift, block, and nuisance terms are projected out as confounds.
prewhiten	Optional prewhitening options (see ?lss).
prepass	Optional list forwarded to sbhm_prepass().
match	Optional list forwarded to sbhm_match().
oasis	Optional list forwarded to lss(..., method = "oasis"). K defaults to ncol(sbhm$B).
amplitude	Amplitude options (see ?lss_sbhm).
return	One of "amplitude", "coefficients", or "both".
validate	Logical; when TRUE, performs basic checks (sampling frame compatibility, temporal alignment) analogous to lss_design().
...	Reserved for future use.

Details

This function wraps lss_sbhm() by converting the event_model into an OASIS design_spec that uses the SBHM basis HRF, and by mapping baseline_model terms to nuisance regressors for projection.

Value

Same return contract as lss_sbhm().

Examples

```
## Not run:
  library(fmridesign)
  sframe <- fmrihrf::sampling_frame(blocklens = c(150, 150), TR = 2)
  trials <- data.frame(onset = c(10,30,50, 10,30,50), run = rep(1:2, each=3))
  emod <- event_model(onset ~ trialwise(basis = "spmg1"), data = trials,
                      block = ~run, sampling_frame = sframe)
  Y <- matrix(rnorm(300*100), 300, 100)
  out <- lss_sbhm_design(Y, sbhm, emod)
```

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

`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
)
```

Arguments

`Y`	Numeric matrix of BOLD data (time x voxels).
`events`	Data frame with `onset`, `duration` and `condition` columns.
`hrf_estimates`	A VoxelHRF object returned by `estimate_voxel_hrf`.
`nuisance_regs`	Optional numeric matrix of nuisance regressors.
`engine`	Computational engine: "R" for pure R implementation (default), "C++" for optimized C++ (experimental).
`chunk_size`	Number of voxels to process per batch (C++ engine only).
`verbose`	Logical; display progress bar.
`backing_dir`	Directory for bigmemory backing files. If NULL, a temporary directory is used (C++ engine only).

Value

An object of class LSSBeta for C++ engine, or a numeric matrix (n_trials x n_vox) for R engine.

Examples

```
## Not run:
set.seed(1)
Y <- matrix(rnorm(100), 50, 2)
events <- data.frame(onset = c(5, 25), duration = 1,
                     condition = "A")
basis <- fmrihrf::hrf_gamma()
sframe <- fmrihrf::sampling_frame(blocklens = nrow(Y), TR = 1)
times <- fmrihrf::samples(sframe, global = TRUE)
rset <- fmrihrf::regressor_set(onsets = events$onset,
                               fac = factor(1:nrow(events)),
                               hrf = basis, duration = events$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)
betas <- lss_with_hrf(Y, events, est, verbose = FALSE, engine = "R")
dim(betas)

## End(Not run)
```

LSSBeta *LSSBeta object*

Description

Simple list-based S3 class returned by `lss_with_hrf` containing trial-wise beta estimates.

Value

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

Examples

```
## Not run:
Y <- matrix(rnorm(100), 50, 2)
events <- data.frame(onset = c(5, 25), duration = 1, condition = "A")
basis <- fmrihrf::HRF_SPMG1
est <- estimate_voxel_hrf(Y, events, basis)
fit <- lss_with_hrf(Y, events, est, engine = "C++", verbose = FALSE)
class(fit)

## End(Not run)
```

mixed_precompute *Precompute Workspace for Optimized Mixed Model*

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 × p)
Z	Random effects design matrix (n × q)
K	Kinship/covariance matrix for random effects (q × 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 *Mixed Model Solver*

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* + *Z*u + error, where u ~ N(0, Vu*K) and error ~ N(0, Ve*I). 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

```
## Not run:
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)

## End(Not run)
```

---

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 × p)
Z	Random effects design matrix (n × q)
Y	Response data - can be a vector (single voxel) or matrix (n × V for multiple voxels)

K	Kinship/covariance matrix for random effects (q × 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 *Construct OASIS options*

Description

Convenience constructor for the `oasis=` list accepted by `lss(method="oasis")`. Unknown fields are allowed via `...` for forward compatibility.

Usage

```
oasis_options(
  design_spec = NULL,
  K = 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

design_spec	Optional design spec list used to build X via fmrihrf.
K	Optional basis dimension override.
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.
...	Additional options.

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 *Plot HRF Recovery Comparison*

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

```
## Not run:
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)

## End(Not run)
```

---

prewhiten_options *Construct prewhitening options*

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
)
```

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 run identifiers.
`parcels`	Optional parcel ids.
`exact_first`	`"ar1"` or `"none"`.
`compute_residuals`	Logical.

Value

A list with class "fmrilss_prewhiten_options".

Examples

```
prewhiten_options(method = "ar", p = 1, pooling = "run")
```

---

project_confounds *Project Out Confound Variables*

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 x 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. The resulting matrix Q can be applied to data to remove the influence of confound regressors.

Value

Projection matrix Q (n x n) that projects out the column space of X

Examples

```
## Not run:
n <- 100
X_confounds <- cbind(1, 1:n)

Q <- project_confounds(X_confounds)

Y_clean <- Q %*% Y_raw

## End(Not run)
```

`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

`X_confounds`	Confound design matrix (n x k)
`Y_data`	Data matrix (n x V) where V is number of voxels
`C_trials`	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

```
## Not run:
n <- 200; k <- 5; V <- 1000; T <- 50
X_confounds <- cbind(1, 1:n, rnorm(n*3))
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)

## End(Not run)
```

sbhm_build *Build a Shared-Basis HRF Library (SBHM)*

Description

Learn a low-rank shared time basis from a parameterized HRF library using `fmrihrf::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 = 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", "spmg1")
)
```

Arguments

`library_spec`	Either NULL (when `library_H` is provided) or a list with: • `fun`: a function compatible with `fmrihrf::hrf_library(fun, pgrid, ...)` that returns an `fmrihrf` HRF object when called with parameters. • `pgrid`: a data.frame of parameter combinations (see examples). • `span`: numeric, HRF span in seconds (default span). • `precision`: numeric, evaluation precision (default 0.1 sec). • `method`: evaluation method for `fmrihrf::evaluate()` (default "conv"). • `extras`: optional list of additional arguments passed to `hrf_library`.
`library_H`	Optional precomputed TxK matrix of candidate HRFs, already aligned to the TR grid `tgrid` (or `sframe`). Mutually exclusive with `library_spec`.
`r`	Target rank for the shared basis (default 6). Clipped to `min(T, K)`.
`sframe`	Optional `fmrihrf::sampling_frame`, 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 "spmg1". If "spmg1", 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: list with r, K, normalize, baseline

Examples

```
## Not run:
  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)

## End(Not run)
```

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

Description

Convert a shared time basis matrix B (T×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

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

Value

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

Examples

```
## Not run:
  hrf_B <- sbhm_hrf(sbhm$B, sbhm$tgrid, sbhm$span)

## End(Not run)
```

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

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

Value

A list with:

- idx length-V integer indices of best-matching library HRF (1..K)
- margin length-V numeric: score(top1) - score(top2)
- alpha_hat r×V matrix: the selected library coordinates (unwhitened, unnormalized)
- scores optional K×V cosine score matrix (returned when topK > 1)
- weights optional top-K weights per voxel (when topK > 1)

Examples

```
## Not run:
  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

## End(Not run)
```

`sbhm_prepass` *SBHM Prepass: Aggregate Fit in Shared Basis*

Description

Compute per-voxel coefficients in the shared SBHM basis by fitting a single aggregate GLM with one regressor per basis column (trials summed), optionally residualizing by nuisances and prewhitening. This produces `beta_bar` (r×V) that you can feed to `sbhm_match()`.

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

`Y`	Numeric matrix T×V of fMRI time series.
`sbhm`	SBHM object from `sbhm_build()` (must contain B, S, A, tgrid, span).
`design_spec`	List describing events (same shape as `oasis$design_spec`). Must contain `sframe` and cond with `onsets` (and optional `duration`, `amplitude`, `span`). `cond$hrf` is ignored and replaced with `sbhm_hrf`. Optional `others` (list of other conditions) will be aggregated as nuisances.
`Nuisance`	Optional T×P nuisance matrix (motion, drift, etc.).

`prewhiten` Optional fmriAR prewhitening options (see `?lss`). If provided, Y and design are prewhitened together.

`ridge` Optional list for targeted ridge shrinkage in the prepass solve:

- `mode`: "fractional" (default) or "absolute". Fractional scales by mean(diag(G)).
- `lambda`: nonnegative scalar (default 0.01 in fractional mode).
- `alpha_ref`: r-vector to shrink towards (default zero vector).

`data_fac` Optional list for external factorization: `scores` (T×q), `loadings` (q×V). If provided, computes X'Y via (X'Scores) × Loadings. In this PR2 version, prewhitening is not applied when `data_fac` is used.

Details

Notes:

- Aggregated per-basis regressors can be highly collinear, making G = A' A ill-conditioned. A small ridge is recommended for stability. The default uses fractional mode with `lambda = 0.01` (scaled by mean(diag(G))).
- When `data_fac` is provided (factorized data path), prewhitening is skipped in this version; both dense and factorized paths perform nuisance residualization consistently.

Value

List with:

- `beta_bar` r×V aggregate coefficients
- `A_agg` T×r aggregated per-basis design (after any residualization/whitening)
- `G` r×r crossprod of A_agg
- `diag` list with K=r, ntrials, times, used_prewhiten

Examples

```
## Not run:
  library(fmrihrf)
  set.seed(1)
  Tlen <- 120; V <- 5; r <- 4
  sframe <- sampling_frame(blocklens = Tlen, TR = 1)
  H <- cbind(exp(-seq(0, 30, length.out = Tlen)/4),
             exp(-seq(0, 30, length.out = Tlen)/6))
  sbhm <- sbhm_build(library_H = H, r = r, sframe = sframe, normalize = TRUE)
  onsets <- seq(5, 95, by = 10)
 design_spec <- list(sframe = sframe, cond = list(onsets = onsets, duration = 0, span = 30))
  hrf_B <- sbhm_hrf(sbhm$B, sbhm$tgrid, sbhm$span)
 rr <- fmrihrf::regressor(onsets = onsets, hrf = hrf_B, duration = 0, span = 30, summate = FALSE)
  X <- fmrihrf::evaluate(rr, grid = sbhm$tgrid, precision = 0.1, method = "conv")
  betas_true <- matrix(rnorm(r), r)
  Y <- matrix(rnorm(Tlen*V, sd = 0.5), Tlen, V)
  Y[,1] <- Y[,1] + X %*% betas_true
  pre <- sbhm_prepass(Y, sbhm, design_spec)
  str(pre)
```

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

---

sbhm_project *Project Trial-wise SBHM Coefficients to Scalar Amplitudes*

Description

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

Usage

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

Arguments

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

Value

Numeric matrix ntrials x V of scalar amplitudes.

Examples

```
## Not run:
  r <- nrow(alpha_hat)
  ntrials <- nrow(beta_mat) / r
  beta_rt <- array(beta_mat, dim = c(r, ntrials, ncol(beta_mat)))
  amps <- sbhm_project(beta_rt, alpha_hat)

## End(Not run)
```

stglmnet_options *Construct stglmnet backend options*

Description

Convenience constructor for the `stglmnet=` list accepted by `lss(method = "stglmnet")`. Unknown fields are allowed via `...` for forward compatibility.

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).
`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 `prewhiten=` 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, lss(method="stglmnet") returns a list containing beta, fit metadata, and the selected lambda.

...
: Additional backend options.

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().

Examples

```
## Not run:
Y <- matrix(rnorm(100), 50, 2)
events <- data.frame(onset = c(5, 25), duration = 1, condition = "A")
basis <- fmrihrf::HRF_SPMG1
est <- estimate_voxel_hrf(Y, events, basis)
class(est)

## End(Not run)
```

Index

benchmark_mixed_solve, 3

calculate_recovery_metrics, 4
compare_hrf_recovery, 4
create_lwu_grid, 5

estimate_voxel_hrf, 6

fit_oasis_grid, 7
fmrilss_options, 8

generate_lwu_data, 8
generate_rapid_design, 9

item_build_design, 10
item_compute_u, 11
item_cv, 13
item_fit, 14
item_from_lsa, 15
item_predict, 17
item_slice_fold, 17

lsa, 18
lss, 19, *27–30*
lss_beta_cpp, 24
lss_cpp_optimized, 25
lss_design, 26
lss_fit_wrappers, 29
lss_naive, 29
lss_naive_fit, 31
lss_optimized, 32
lss_optimized_fit, 33
lss_sbhm, 34
lss_sbhm_design, 36
lss_sbhm_design(), *35*
lss_with_hrf, 38
LSSBeta, *38*, 39

mixed_precompute, 40
mixed_solve, 40
mixed_solve_cpp *(*mixed_solve*)*, 40
mixed_solve_optimized, 42

oasis_options, *20*, *27*, 43

plot_hrf_comparison, 44
prewhiten_options, *20*, *21*, *27*, 45
project_confounds, 46
project_confounds_cpp, 47

sbhm_build, 48
sbhm_hrf, 50
sbhm_match, 50
sbhm_match(), *35*
sbhm_prepass, 52
sbhm_prepass(), *35*
sbhm_project, 54
stglmnet_options, *20*, 55

VoxelHRF, *6*, *38*, 56