

Package: fmriAR (via r-universe)

October 11, 2026

Type Package

Title Fast AR and ARMA Noise Whitening for Functional MRI (fMRI)
Design and Data

Version 0.4.2

Description Lightweight utilities to estimate autoregressive (AR) and autoregressive moving average (ARMA) noise models from residuals and apply matched generalized least squares to whiten functional magnetic resonance imaging (fMRI) design and data matrices. The ARMA estimator follows a classic 1982 approach [doi:10.1093/biomet/69.1.81](https://doi.org/10.1093/biomet/69.1.81), and a restricted AR family mirrors workflows described by Cox (2012) [doi:10.1016/j.neuroimage.2011.08.056](https://doi.org/10.1016/j.neuroimage.2011.08.056).

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.0)

Imports Rcpp, stats, utils

Suggests testthat (>= 3.0.0), knitr, rmarkdown, abind, withr

LinkingTo Rcpp, RcppArmadillo

SystemRequirements OpenMP (optional)

Config/testthat/edition 3

VignetteBuilder knitr

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

BugReports <https://github.com/bbuchsbaum/fmriAR/issues>

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

Date/Publication 2026-10-11 03:21:07 UTC

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

RemoteRef HEAD

RemoteSha bb0789c7d1e720aab318965519ac0f343e4419fe

Contents

acorr_diagnostics	2
acvf_bias_matrix	3
afni_restricted_plan	4
fit_noise	5
noise_acvf	8
print.fmriAR_plan	10
sandwich_from_whitened_resid	11
whiten	12
whiten_apply	13
Index	15

acorr_diagnostics	<i>Autocorrelation diagnostics for residuals</i>
-------------------	--

Description

Autocorrelation diagnostics for residuals

Usage

```
acorr_diagnostics(  
  resid,  
  runs = NULL,  
  max_lag = 20L,  
  aggregate = c("mean", "median", "none")  
)
```

Arguments

resid	Numeric matrix (time x voxels), typically whitened residuals.
runs	Optional run labels, length nrow(resid). When supplied, each run is centred separately and no lag product spans a run boundary.
max_lag	Maximum lag to evaluate.
aggregate	How per-voxel autocorrelations are combined: "mean", "median" (across voxels, lag by lag), or "none" (lags x voxels matrix).

Value

List of autocorrelation values and nominal confidence interval.

Examples

```
# Generate example residuals with some autocorrelation
n_time <- 200
n_voxels <- 50
resid <- matrix(rnorm(n_time * n_voxels), n_time, n_voxels)

# Add some AR(1) structure
for (v in 1:n_voxels) {
  resid[, v] <- filter(resid[, v], filter = 0.3, method = "recursive")
}

# Check autocorrelation
acorr_check <- acorr_diagnostics(resid, max_lag = 10, aggregate = "mean")

# Examine lag-1 autocorrelation (lag 0 is not returned, so acf[1] is lag 1)
lag1_acorr <- acorr_check$acf[1]
```

acvf_bias_matrix

Bias matrix relating raw residual autocovariance to the truth

Description

Builds the linear map A satisfying $E[\gamma_{\text{raw}}] = A \gamma_{\text{true}}$ induced by projecting a design out of the data. Autocovariance estimated from GLM residuals is biased low, increasingly so for richer designs, because $E[\hat{e} \hat{e}'] = M \Sigma M$ for the residual-forming projection M . Solving $A \gamma = \gamma_{\text{raw}}$ removes that bias.

Usage

```
acvf_bias_matrix(design, runs = NULL, censor = NULL, max_lag = 25L)
```

Arguments

design	Numeric design matrix (timepoints x regressors), the one whose projection produced the residuals.
runs	Optional run labels, length <code>nrow(design)</code> . Each label must occupy one contiguous block and may not be missing. Lag products never cross a run boundary.
censor	Optional indices of censored timepoints (1-based), excluded exactly as in fit_noise() .
max_lag	Highest lag to correct.

Details

Ordinarily you do not need this: pass design to [fit_noise\(\)](#) and it builds and applies the matrices itself. Use this directly to inspect the bias, or to cache the matrices when many datasets share one design and segmentation and feed them back as `acvf_correction`.

One matrix is returned per run, because [fit_noise\(\)](#) estimates each run separately and the residual operator restricted to a run still involves the whole design. Under pooling = "parcel", which

pools lag products across runs, the run matrices are combined in proportion to the lag pairs each run contributes.

Projecting out run means and slow drift leaves the map with near-null directions (roughly a constant offset across lags). When the correction is applied, those directions are fixed by the same short-memory assumption the truncated system already makes (the autocovariance is near zero at the end of the budget) rather than taken from the data, which keeps the correction stable under censoring.

The correction is exact for noise whose autocovariance dies within `max_lag`. Under long memory it is partial, since truncating the system aliases in structure past the budget. With high-pass filtering (DCT, 128 s) a budget near 20-25 lags is typically enough; without high-pass it exceeds 150 and the approach stops being practical.

Cost is $O(\text{max_lag}^2 * \text{n_valid} * r)$ per run for a design of rank r .

Value

A named list of $(\text{max_lag} + 1)$ square matrices, one per run, rows indexed by the lag of the raw estimate and columns by the lag of the truth.

See Also

`fit_noise()`, `noise_acvf()`

Examples

```
n <- 120
X <- cbind(1, poly(seq_len(n), 3))
A <- acvf_bias_matrix(X, max_lag = 5)
round(A[[1]][1:3, 1:3], 3)
```

`afni_restricted_plan` *Build an AFNI-style restricted AR plan from root parameters*

Description

Build an AFNI-style restricted AR plan from root parameters

Usage

```
afni_restricted_plan(
  resid,
  runs = NULL,
  parcels = NULL,
  p = 3L,
  roots,
  estimate_ma1 = TRUE,
  exact_first = TRUE
)
```

Arguments

resid	(n x v) residual matrix (used only if estimate_ma1=TRUE)
runs	integer vector length n (optional)
parcels	integer vector length v (optional; if provided, plan pooling='parcel')
p	either 3 or 5
roots	either a single list with elements named as needed - for p=3: list(a, r1, t1, vrt = 1.0) - for p=5: list(a, r1, t1, r2, t2, vrt = 1.0) or a named list of such lists keyed by parcel id (character) for per-parcel specs. As in AFNI, a, r1, r2 must be non-negative (values above 0.95 are clamped to 0.95) and angles are clamped to $[0, \pi]$. vrt is AFNI's signal-to-total variance ratio $s^2 / (s^2 + w^2)$ for additive white noise w; it is used when estimate_ma1 = FALSE.
estimate_ma1	logical. If TRUE, estimate an MA(1) term from the AR-filtered voxels to stand in for AFNI's additive white noise (vrt is then ignored). If FALSE, the white-noise share is taken from vrt: for $vrt < 1$ the plan is the exact ARMA(p, p) equivalent of AR(p) plus white noise, so its autocorrelation is vrt times the AR autocorrelation at every non-zero lag; vrt = 1 (default) gives the pure AR(p).
exact_first	apply exact AR(1) scaling at segment starts (harmless here; default TRUE)

Value

An fmriAR_plan with method = "afni" that can be supplied to [whiten_apply\(\)](#).

Examples

```
NULL
```

fit_noise	<i>Fit an AR/ARMA noise model (run-aware) and return a whitening plan</i>
-----------	---

Description

Fit an AR/ARMA noise model (run-aware) and return a whitening plan

Usage

```
fit_noise(
  resid = NULL,
  Y = NULL,
  X = NULL,
  runs = NULL,
  censor = NULL,
  method = c("ar", "arma"),
  p = "auto",
  q = 0L,
  p_max = 6L,
```

```

q_max = 2L,
exact_first = c("ar1", "none"),
pooling = c("global", "run", "parcel"),
parcels = NULL,
parcel_sets = NULL,
multiscale = c("pacf_weighted", "acvf_pooled"),
ms_mode = NULL,
p_target = NULL,
beta = 0.5,
hr_iter = 0L,
step1 = c("burg", "yw"),
design = NULL,
acvf_correction = NULL,
correction_max_lag = 25L,
parallel = FALSE
)

```

Arguments

resid	Numeric matrix (time x voxels) of residuals from an initial OLS fit.
Y	Optional data matrix used to compute residuals when resid is omitted.
X	Optional design matrix used with Y to compute residuals.
runs	Optional run labels, one per timepoint. Each label must occupy one contiguous block and may not be missing.
censor	Optional integer vector of 1-based timepoint indices to exclude from AR parameter estimation, or a logical vector of length nrow(resid) where TRUE indicates censored timepoints. Censored frames (e.g., motion-corrupted) are excluded when computing autocorrelations. Each run's estimation uses only its own valid (non-censored) segments.
method	Either "ar" or "arma". ARMA models are estimated by a Hannan–Rissanen regression pooled over voxels (every voxel is treated as a replicate series sharing the coefficients), with all lags kept inside contiguous run/censor segments.
p	AR order (integer or "auto" if method == "ar"). An explicit order is honoured even when it exceeds p_max.
q	MA order (integer), or "auto" (ARMA only) to choose it by BIC from 0:q_max.
p_max	Maximum AR order when p = "auto". The order is chosen by BIC whose sample size is the number of frames times the effective number of independent voxels (a design effect from the mean inter-voxel correlation), so pooling many independent voxels can support a higher order than one series would. For method = "arma", p = "auto" searches 0:p_max jointly with the MA order by BIC on the pooled Hannan–Rissanen regression.
q_max	Maximum MA order when q = "auto".
exact_first	How <code>whiten_apply()</code> treats the start of each run or post-censoring segment. "ar1" (default; the name is historical) applies the exact stationary initialisation of the fitted AR/ARMA model, i.e. exact GLS whitening within each segment; for AR(1) this is the familiar $\sqrt{1 - \phi^2}$ scaling of the first sample. "none" uses the conditional filter with a zero pre-sample.

pooling	Combine parameters across runs or parcels ("global", "run", "parcel").
parcels	Integer vector (length = ncol(resid)) giving fine parcel memberships when pooling = "parcel".
parcel_sets	Optional named list with entries coarse, medium, fine of equal length specifying nested parcel labels for multi-scale pooling.
multiscale	Multi-scale pooling mode when parcel_sets is supplied ("pacf_weighted" or "acvf_pooled"), or TRUE/FALSE to toggle pooling.
ms_mode	Explicit multiscale mode when multiscale is logical.
p_target	Target AR order for multi-scale pooling (defaults to p_max).
beta	Size exponent for multi-scale weights (default 0.5).
hr_iter	Number of Hannan–Rissanen refinement iterations for ARMA.
step1	Preliminary high-order AR fit method for HR ("burg" or "yw").
design	Optional design matrix (timepoints x regressors) whose projection produced resid. Supplying it corrects the downward bias that projecting a design out of the data puts into the autocovariance, and hence into phi. Opt-in, because it changes estimates and needs the design to be the one that actually formed the OLS residuals. A numerical orthogonality check rejects detectable mismatches, including typical raw, GLS, robust, or unrelated residuals. Orthogonality cannot prove the exact residual-forming provenance, so the caller remains responsible for supplying the matching design. Supported for every pooling mode with method = "ar"; under pooling = "parcel" the per-run bias maps are combined in proportion to the lag pairs each run contributes.
acvf_correction	Precomputed bias matrices from <code>acvf_bias_matrix()</code> , as an alternative to design when many datasets share one design. A single matrix is applied to every run; a list is matched against the runs in order. Mutually exclusive with design.
correction_max_lag	Lag budget for the bias correction (default 25). The correction solves a system truncated at this lag, so too small a budget leaves bias behind; too large a one approaches the run length and makes the system ill-conditioned, which is refused with a warning rather than solved. A design leaving fewer residual degrees of freedom than the budget also cannot support it, and the budget is reduced accordingly, again with a warning. Ignored unless design is supplied.
parallel	Reserved for future parallel estimation (logical).

Value

An object of class `fmriAR_plan` used by `whiten_apply()`. Besides the AR/MA coefficients the plan carries the noise scale and shape it was fitted from, so consumers can reconstruct the covariance it implies rather than only its correlation structure:

- `gamma`: list of autocovariance vectors, one per pooling unit – a single entry for pooling = "global", one per run for pooling = "run". Lags run 0 to the highest the data supported, which is governed by `p_max` and the run length rather than by `p`, so `fit_noise(p = 1, p_max = 6)` returns seven values, not two. Under global pooling every run is truncated to the shortest

available length before averaging, since a zero-padded autocovariance is not a valid covariance. Global AR coefficients are fitted by Yule–Walker on the frame-weighted average of the runs’ autocorrelations.

- `sigma2`: list of innovation variances, matching `gamma`. For `method = "ar"` it is derived as $\gamma_0 - \sum_k \phi_k \gamma_k$ from the coefficients stored on the plan so the two are always mutually consistent, and is NA whenever `gamma` does not reach lag `length(phi)` – heavy censoring can truncate it that far, and a partial sum would overstate the innovation variance rather than report that it is unavailable. For `method = "arma"` it is the mean squared exact-whitened residual over all voxels and valid frames.
- `gamma_by_parcel`, `sigma2_by_parcel`: the same quantities per parcel when `pooling = "parcel"`, keyed like `phi_by_parcel`.

For a run-stationary noise process with autocovariance `gamma`, the covariance of the data within a run is the Toeplitz matrix built from it, which is what makes design-specific variance calculations possible downstream without refitting.

Examples

```
# Generate example data with AR(1) structure
n_time <- 200
n_voxels <- 50
phi_true <- 0.5

# Simulate residuals with AR(1) structure
resid <- matrix(0, n_time, n_voxels)
for (v in 1:n_voxels) {
  e <- rnorm(n_time)
  resid[1, v] <- e[1]
  for (t in 2:n_time) {
    resid[t, v] <- phi_true * resid[t-1, v] + e[t]
  }
}

# Fit AR model
plan <- fit_noise(resid, method = "ar", p = 1)

# With multiple runs
runs <- rep(1:2, each = 100)
plan_runs <- fit_noise(resid, runs = runs, method = "ar", pooling = "run")
```

noise_acvf

Run- and censor-aware noise autocovariance

Description

Estimates the autocovariance of fMRI noise directly, without fitting an AR model first. Lag products never cross a run boundary or a censoring gap, and the mean is removed per run rather than per fragment, so scrubbed data does not have its autocorrelation destroyed by the centering.

Usage

```
noise_acvf(
  resid,
  runs = NULL,
  censor = NULL,
  max_lag = 20L,
  pooling = c("global", "run", "parcel"),
  parcels = NULL,
  design = NULL,
  correction_max_lag = 25L
)
```

Arguments

<code>resid</code>	Numeric matrix of residuals (timepoints x voxels).
<code>runs</code>	Optional run labels, length <code>nrow(resid)</code> . Each label must occupy one contiguous block and may not be missing.
<code>censor</code>	Optional indices of censored timepoints (1-based).
<code>max_lag</code>	Highest lag to return.
<code>pooling</code>	"global" for one autocovariance over all runs, "run" for one per run, "parcel" for one per parcel.
<code>parcels</code>	Parcel labels (length <code>ncol(resid)</code>), required when <code>pooling = "parcel"</code> .
<code>design</code>	Optional design matrix whose projection produced <code>resid</code> . The residuals must be ordinary least-squares residuals numerically orthogonal to this design. A numerical check rejects detectable mismatches, but orthogonality alone cannot prove the residual-forming provenance; the caller must supply the matching design. When supplied the residual bias is corrected; see acvf_bias_matrix() .
<code>correction_max_lag</code>	Lag budget used to estimate and undo residual projection bias when design is supplied. This is independent of <code>max_lag</code> : correction needs enough tail information even when only a few output lags are requested.

Details

This is the estimator [fit_noise\(\)](#) uses internally, exposed so consumers can get the covariance itself. For a run-stationary process with autocovariance γ , the covariance of any linear contrast L factors as $\text{sum}_h \gamma_h * L' T_h L'$ with T_h the lag- h indicator, so γ plus a design is enough to compute exact contrast variances without refitting.

Unlike [acorr_diagnostics\(\)](#), which returns normalized autocorrelation for inspection, this returns covariances on the scale of the data.

Value

An object of class `fmriAR_acvf`: a list with

- `acvf`: named list of autocovariance vectors, lags 0 upward.

- `pairs`: matching list of the pair count behind each lag, so a consumer can tell an estimate backed by thousands of pairs from one backed by three.
- `n_segments`, `segment_lengths`: the segmentation actually used.
- `max_lag`, `pooling`, `corrected`.

Lags with no surviving pairs are dropped rather than reported as zero.

See Also

`fit_noise()`, `acvf_bias_matrix()`

Examples

```
set.seed(1)
r <- matrix(rnorm(400 * 5), 400, 5)
a <- noise_acvf(r, runs = rep(1:2, each = 200), max_lag = 5, pooling = "run")
a$acvf[["1"]]
```

<code>print.fmriAR_plan</code>	<i>Pretty-print an fmriAR whitening plan</i>
--------------------------------	--

Description

Pretty-print an fmriAR whitening plan

Usage

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

Arguments

<code>x</code>	An object returned by <code>fit_noise()</code> .
<code>...</code>	Unused; included for S3 compatibility.

Value

The input plan, invisibly.

Examples

```
resid <- matrix(rnorm(60), 20, 3)
plan <- fit_noise(resid, method = "ar", p = 2)
print(plan)
```

sandwich_from_whitened_resid

GLS standard errors from whitened residuals

Description

GLS standard errors from whitened residuals

Usage

```
sandwich_from_whitened_resid(
  Xw,
  Yw,
  beta = NULL,
  type = c("iid", "hc0", "hac"),
  df_mode = c("rankX", "n-p"),
  runs = NULL,
  hac_lag = NULL
)
```

Arguments

Xw	Whitened design matrix.
Yw	Whitened data matrix (time x voxels).
beta	Optional coefficients (p x v); estimated if NULL.
type	"iid" (default) assumes the whitened errors are white and homoskedastic. "hc0" is a heteroskedasticity-robust sandwich. "hac" is a Newey-West (Bartlett kernel) sandwich computed within runs, which stays valid when the noise model leaves some autocorrelation behind.
df_mode	Degrees-of-freedom mode: "rankX" (default) or "n-p".
runs	Optional run labels (length nrow(Xw)). Used by type = "hac" so that no lag product spans a run boundary.
hac_lag	Bandwidth for type = "hac". Defaults to $\text{floor}(4 * (n_{\text{run}} / 100)^{(2/9)})$ using the median run length.

Value

List containing standard errors, innovation variances, and XtX inverse. Coefficients that are not estimable because Xw is rank deficient get NA standard errors (and NA rows/columns in XtX_inv), following `stats::lm()`.

Examples

```
# Generate example whitened data
n_time <- 200
n_pred <- 3
n_voxels <- 50
Xw <- matrix(rnorm(n_time * n_pred), n_time, n_pred)
Yw <- matrix(rnorm(n_time * n_voxels), n_time, n_voxels)

# Compute standard errors
se_result <- sandwich_from_whitened_resid(Xw, Yw, type = "iid")

# Extract standard errors for first voxel
se_voxel1 <- se_result$se[, 1]
```

whiten	<i>Fit and apply whitening in one call</i>
--------	--

Description

Fit and apply whitening in one call

Usage

```
whiten(X, Y, runs = NULL, censor = NULL, ...)
```

Arguments

X	Design matrix (time x regressors).
Y	Data matrix (time x voxels).
runs	Optional run labels.
censor	Optional censor indices.
...	Additional parameters passed to fit_noise() .

Value

List with whitened X and Y matrices.

Examples

```
# Create example data
n_time <- 200
n_pred <- 3
n_voxels <- 50
X <- matrix(rnorm(n_time * n_pred), n_time, n_pred)
Y <- X %*% matrix(rnorm(n_pred * n_voxels), n_pred, n_voxels) +
  matrix(rnorm(n_time * n_voxels, sd = 2), n_time, n_voxels)

# One-step whitening
whitened <- whiten(X, Y, method = "ar", p = 2)
```

whiten_apply	<i>Apply a whitening plan to design and data matrices</i>
--------------	---

Description

Apply a whitening plan to design and data matrices

Usage

```
whiten_apply(
  plan,
  X,
  Y,
  runs = NULL,
  run_starts = NULL,
  censor = NULL,
  parcels = NULL,
  inplace = FALSE,
  parallel = TRUE
)
```

Arguments

plan	Whitening plan from <code>fit_noise()</code> .
X	Numeric matrix of predictors (time x regressors).
Y	Numeric matrix of data (time x voxels).
runs	Optional run labels, one per row. Each label must occupy one contiguous block and may not be missing.
run_starts	Optional 0-based run start indices (alternative to runs).
censor	Optional censored timepoints: 1-based indices or a logical mask of length <code>nrow(Y)</code> . The filter restarts (with exact stationary initialisation when the plan uses <code>exact_first</code>) at the first frame after each censored one. When omitted and the plan was fitted on data with the same number of timepoints, the plan's own censor set is used.
parcels	Optional parcel labels (length = <code>ncol(Y)</code>) when using parcel plans.
inplace	Deprecated and ignored. The inputs were never modified (R's copy-on-modify semantics prevent it safely), so TRUE only made the result invisible. Supplying it now warns; it will be removed in a future release.
parallel	Use OpenMP parallelism if available. The thread count can be capped with <code>options(fMRIAR.max_threads = k)</code> .

Value

List with whitened data. Parcel plans return `X_by` per parcel; others return a single `X` matrix. When censoring is in effect the list also carries `censor`, the censored row indices. Those rows are still present in `X/Y` (filtered from the preceding history) and should be dropped before fitting, e.g. `lm.fit(out$X[-out$censor, ], out$Y[-out$censor, ])`.

Examples

```
# Create example design matrix and data
n_time <- 200
n_pred <- 3
n_voxels <- 50
X <- matrix(rnorm(n_time * n_pred), n_time, n_pred)
Y <- X %%% matrix(rnorm(n_pred * n_voxels), n_pred, n_voxels) +
      matrix(rnorm(n_time * n_voxels), n_time, n_voxels)

# Fit noise model from residuals
residuals <- Y - X %%% solve(crossprod(X), crossprod(X, Y))
plan <- fit_noise(residuals, method = "ar", p = 2)

# Apply whitening
whitened <- whiten_apply(plan, X, Y)
Xw <- whitened$X
Yw <- whitened$Y
```

Index

`acorr_diagnostics`, [2](#)
`acorr_diagnostics()`, [9](#)
`acvf_bias_matrix`, [3](#)
`acvf_bias_matrix()`, [7](#), [9](#), [10](#)
`afni_restricted_plan`, [4](#)

`fit_noise`, [5](#)
`fit_noise()`, [3](#), [4](#), [9](#), [10](#), [12](#), [13](#)

`noise_acvf`, [8](#)
`noise_acvf()`, [4](#)

`print_fmriAR_plan`, [10](#)

`sandwich_from_whitened_resid`, [11](#)
`stats::lm()`, [11](#)

`whiten`, [12](#)
`whiten_apply`, [13](#)
`whiten_apply()`, [5–7](#)