Package: fmriAR (via r-universe)

June 14, 2026

Type Package

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

Version 0.3.2

Author Bradley Buchsbaum [aut, cre]

Maintainer Bradley Buchsbaum <brad.buchsbaum@gmail.com>

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>, and a restricted AR family mirrors workflows described by Cox (2012) <doi:10.1016/j.neuroimage.2011.08.056>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.0)

Imports Rcpp, stats

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/

NeedsCompilation yes

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

Date/Publication 2026-04-15 15:41:46 UTC

RemoteUrl https://github.com/cran/fmriAR

RemoteRef HEAD

RemoteSha 62840588d0be62a9a7d95ca41f4d771a92832748

Contents

`acorr_diagnostics` *Autocorrelation diagnostics for residuals*

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.
`max_lag`	Maximum lag to evaluate.
`aggregate`	Aggregation across voxels: "mean", "median", or "none".

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
lag1_acorr <- acorr_check$acf[2]  # First element is lag-0 (always 1)
```

`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.
`estimate_ma1`	logical, if TRUE estimate MA(1) on AR residuals to mimic AFNI's additive white
`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` *Fit an AR/ARMA noise model (run-aware) and return a whitening plan*

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,
  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"),
  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 integer vector of run identifiers.
`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".
`p`	AR order (integer or "auto" if method == "ar").

q	MA order (integer).
p_max	Maximum AR order when p = "auto".
exact_first	Apply exact AR(1) scaling at segment starts ("ar1" or "none").
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").
parallel	Reserved for future parallel estimation (logical).

Value

An object of class fmriAR_plan used by whiten_apply().

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")
```

print.fmriAR_plan *Pretty-print an fmriAR whitening plan*

Description

Pretty-print an fmriAR whitening plan

Usage

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

Arguments

x	An object returned by `fit_noise()`.
...	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"),
  df_mode = c("rankX", "n-p"),
  runs = NULL
)
```

Arguments

Xw	Whitened design matrix.
Yw	Whitened data matrix (time x voxels).
beta	Optional coefficients (p x v); estimated if NULL.
type	Either "iid" (default) or "hc0" for a robust sandwich.
df_mode	Degrees-of-freedom mode: "rankX" (default) or "n-p".
runs	Optional run labels (reserved for future per-run scaling).

Value

List containing standard errors, innovation variances, and XtX inverse.

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 *Fit and apply whitening in one call*

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 *Apply a whitening plan to design and data matrices*

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 fit_noise().
X	Numeric matrix of predictors (time x regressors).
Y	Numeric matrix of data (time x voxels).
runs	Optional run labels.
run_starts	Optional 0-based run start indices (alternative to runs).
censor	Optional indices of censored TRs (1-based); filter resets after gaps.
parcels	Optional parcel labels (length = ncol(Y)) when using parcel plans.
inplace	Modify inputs in place (logical).
parallel	Use OpenMP parallelism if available.

Value

List with whitened data. Parcel plans return `X_by` per parcel; others return a single `X` matrix.

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