Package 'fgls'

November 7, 2026

Type Package

Title Feasible GLS for Serial Correlation and Heteroskedasticity

Version 0.1.0

Description Feasible generalized least squares (FGLS) estimators for linear regressions whose errors are serially correlated, heteroskedastic of unknown form, or both. Serial correlation is handled by the autoregressive filter of Perron and González-Coya (2026) <doi:10.1016/j.jeconom.2026.106346>, estimated from a Durbin regression whose order is chosen by an information criterion, with a variance estimator that stays valid when the regressors are only pre-determined rather than strictly exogenous. Heteroskedasticity is handled by the adaptive Lasso estimate of the skedastic function of González-Coya and Perron (2024) <doi:10.1515/jem-2023-0007>, with a bias-corrected sandwich variance. Both corrections can be combined. Benchmark estimators (ordinary least squares with heteroskedasticity and autocorrelation consistent standard errors, Cochrane-Orcutt, infeasible GLS) and the simulation designs of both papers are included.

License MIT + file LICENSE

URL https://github.com/emiliogcs/fgls, https://emiliogcs.com/fgls/

BugReports https://github.com/emiliogcs/fgls/issues

Depends R (>= 4.1.0)

Imports glmnet (>= 4.1), graphics, grDevices, stats, utils

Suggests broom, e1071, generics, jsonlite, knitr, lmtest, parallel, rmarkdown, sandwich, spelling, testthat (>= 3.0.0), tibble, xts, zoo

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.1

NeedsCompilation no

Author Emilio González-Coya [aut, cre, cph] (ORCID:
<https://orcid.org/0009-0001-0362-8843>)

Maintainer Emilio González-Coya <emiliogc@bu.edu>

Repository CRAN

Date/Publication 2026-11-07 15:40:08 UTC

Contents

adf_test . . . 2
compare_fgls . . . 3
fgls . . . 4
fgls-broom . . . 9
fgls-conditions . . . 10
fgls-methods . . . 11
fgls_control . . . 14
lasso_fgls . . . 16
make_dictionary . . . 18
ols_robust . . . 19
plot.fgls . . . 22
serial_fgls . . . 23
simulate_serial . . . 25
summary.fgls . . . 28
test_predetermined . . . 29
weight_diagnostics . . . 31

Index **33**

`adf_test` *Augmented Dickey-Fuller test (screening)*

Description

The regression $\Delta x_t = a + g x_{t-1} + \sum_j c_j \Delta x_{t-j} + e_t$ with the number of lags chosen by BIC (at most $\lfloor 12(T/100)^{1/4} \rfloor$); returns the t statistic on g with MacKinnon's asymptotic critical values. Used by `fgls(check_stationarity = TRUE)` to screen the response and the regressors; it is a screening device, not a replacement for a careful unit-root analysis.

Usage

```
adf_test(x, max_lags = NULL)

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

Arguments

x	A numeric vector (a time series).
max_lags	Largest number of lagged differences.
...	Not used.

Value

A list of class "fgls_adf" with statistic, lags, nobs, critical_values and unit_root_rejected (at 5%). The print method returns x invisibly.

See Also

Other diagnostics: test_predetermined(), weight_diagnostics()

Examples

```
set.seed(1)
adf_test(cumsum(rnorm(200)))$unit_root_rejected
adf_test(rnorm(200))$unit_root_rejected
```

---

compare_fgls *Compare fits side by side*

Description

A table of coefficients, standard errors and confidence-interval widths for several fits, e.g. FGLS against OLS with HAC standard errors (PG26) or OLS with HC3 standard errors (GP24), or the F-C and F-CH stages of a combined fit.

Usage

```
compare_fgls(..., level = 0.95, ols_vcov = NULL)
```

Arguments

...	"fgls" (or plain "lm") objects, optionally named; the names label the columns.
level	Confidence level used for the interval widths.
ols_vcov	Covariance type for lm fits: NULL (automatic), "HAC", "NW" or "HC0"–"HC3".

Details

Plain stats::lm() fits are accepted and converted to OLS with HAC (Andrews quadratic-spectral) standard errors when any other fit corrects for serial correlation, and with HC3 standard errors otherwise; override with ols_vcov. Other models that inherit from "lm" (glm, multivariate mlm fits, weighted fits, fits with an offset, ...) are rejected, because the conversion would replace them by unweighted OLS; use ols_robust() for OLS with robust standard errors.

Value

A data frame with columns `term`, `statistic` (`"estimate"`, `"std.error"`, `"ci.width"`) and one column per fit.

See Also

Other fgls: `fgls()`, `fgls-methods`, `fgls_control()`, `lasso_fgls()`, `plot.fgls()`, `serial_fgls()`, `summary.fgls()`

Examples

```
sim <- simulate_serial(300, seed = 1)
compare_fgls(
  OLS = stats::lm(y ~ x, data = sim),
  FGLS = serial_fgls(y ~ x, data = sim)
)
```

`fgls` *Feasible GLS for serial correlation and/or heteroskedasticity*

Description

`fgls()` fits a linear regression by feasible generalized least squares, correcting for serially correlated errors (Perron and González-Coya, 2026), for heteroskedasticity of unknown form (González-Coya and Perron, 2024), or for both (the "F-CH" estimator of Supplement S-4 of the 2026 paper). `serial_fgls()` and `lasso_fgls()` are shortcuts for the two single corrections.

Usage

```
fgls(
  formula,
  data,
  subset,
  na.action = stats::na.fail,
  serial = c("auto", "fixed", "ma1", "none"),
  hetero = c("lasso", "svr", "wls_s1", "wls_s2", "known", "none"),
 vcov = c("robust", "naive", "nonrobust", "HC0", "HC1", "HC2", "HC3", "bootstrap"),
  z = "auto",
  ar_order = NULL,
  kmax = NULL,
  ic = c("bic", "aic"),
  lagged_dep_vars = NULL,
  wallis_regressor = "auto",
  exog = c("predetermined", "exogenous"),
  first_stage_correction = NULL,
  variance = NULL,
  delta = 0.1,
```

```
  psi_grid = c(0, 0.25, 0.5, 0.75, 1, 2),
  cv_folds = 10L,
  lambda_rule = c("min", "1se"),
  weight_clip = NULL,
  seed = NULL,
  use_t = FALSE,
  time = NULL,
  allow_gaps = FALSE,
  check_stationarity = FALSE,
  control = fgls_control(),
  contrasts = NULL,
  model = TRUE,
  weights,
  ...
)
```

Arguments

formula	A model formula, as in stats::lm(). Include an intercept unless the model truly has none.
data	A data frame, a list, a ts/mts object, or a 'zoo'/'xts' object containing the variables in the formula. Rows must be in time order (unless serial = "none").
subset	Optional expression selecting rows, evaluated in data. For time series without a time index (time, or a ts/zoo data) the selected rows must be an increasing block of consecutive rows (e.g. subset = 51:200): repeated or reversed rows are an error, and so are skipped rows unless allow_gaps = TRUE.
na.action	stats::na.fail() (default) stops on missing values; stats::na.omit() or stats::na.exclude() drop them. For time series only leading and trailing missing values can be dropped. With na.exclude, residuals() and fitted() are padded with NA to the rows selected by subset, as for stats::lm().
serial	Serial-correlation correction: "auto" (AR order by BIC, default), "fixed" (use ar_order), "ma1" (GMA for MA(1) errors; it does not take ar_order, kmax or lagged_dep_vars) or "none".
hetero	Skedastic-function estimator: "lasso" (adaptive Lasso, default), "svr" (support vector regression, requires 'e1071'), "wls_s1"/"wls_s2" (the parametric fits of Romano and Wolf, 2017, on $\log z$ or z), "known" (variances given in variance) or "none". "wls_s1" takes logs, so it needs strictly positive candidate covariates given explicitly in z (e.g. z = "x" for a positive regressor x); the cosine terms of the default z = "auto" are usually negative somewhere, which is an error.
vcov	Variance estimator; the default "robust" is PG26 eq. (29) with the serial correction only, GP24 eqs. (6) and (8) with the heteroskedasticity correction only, and Supplement eqs. (A.4)–(A.5) with both. The types available depend on the corrections: • serial only (hetero = "none"): "robust", "naive" (PG26 eq. 27, exogenous regressors only), "HC0"–"HC3" and "bootstrap" (sieve bootstrap); with serial = "ma1" only "robust";

- heteroskedasticity only (`serial = "none"`): `"robust"`, `"nonrobust"`, `"HC0"`–`"HC3"` and `"bootstrap"` (wild bootstrap);
- both (the default): `"robust"`, `"HC0"`, `"HC3"` and `"bootstrap"` (sieve bootstrap).

Any other choice is an error. See `vcov.fgls()`.

`z` Candidate covariates of the skedastic function: `"auto"` (default) builds $\{x, x^2, \log x, (\log x)^2, \cos x, \cos 2x$ from every non-constant regressor (see `make_dictionary()`); a character vector of regressor names; a one-sided formula evaluated in `data` (e.g. `~ x + I(x^2) + w`); or a numeric matrix / data frame with one row per row of `data`. With `hetero = "wls_s1"` all candidates must be strictly positive, so give them explicitly, e.g. `z = "x"` or `z = ~ x + I(x^2)` for a positive regressor `x`.

`ar_order` Fixed AR order (implies `serial = "fixed"`).

`kmax` Largest AR order considered: `NULL` (default, $\lfloor 12(T/200)^{1/4} \rfloor$), an integer, or `"theory"` ($\lfloor (T/\ln T)^{1/3} \rfloor$). It is capped so that the Durbin regression uses at most half of the sample as regressors.

`ic` Lag-order criterion: `"bic"` (default) or `"aic"`.

`lagged_dep_vars`
Names (or positions) of model-matrix columns that are lags of the dependent variable (not available with `serial = "ma1"`).

`wallis_regressor`
Regressor used for the Wallis recovery of the AR coefficients: `"auto"` (largest Durbin $|t|$), `"median"`, or a name.

`exog` Assumption on the regressors: `"predetermined"` (default) or `"exogenous"`. Estimates do not depend on it; it sets the default of `first_stage_correction` and the notes of the summary.

`first_stage_correction`
Add the first-stage (Durbin) term of PG26 eq. (29) to the variance of the combined estimator. Default `TRUE` when `exog = "predetermined"`.

`variance` Known error variances for `hetero = "known"`: a numeric vector with one value per row of `data`, whose values are selected by `subset` and `na.action` together with the rows of the data (a vector with one value per observation used is also accepted); or a function of the model matrix returning one value per observation used.

`delta` Floor in $\log \max(\hat{u}_t^2, \delta^2)$, in the units of the response (default 0.1, as in the paper).

`psi_grid` Exponents ψ of the adaptive weights; 0 is the plain Lasso.

`cv_folds` Number of cross-validation folds (default 10).

`lambda_rule` Penalty at the minimum of the cross-validation error (`"min"`, default) or the largest penalty within one standard error of it (`"1se"`).

`weight_clip` Optional quantiles `c(lo, hi)` at which the fitted log-variance is winsorized, e.g. `c(0.01, 0.99)`.

`seed` Optional seed for the cross-validation folds and the bootstrap. The session's random number generator state is restored afterwards.

`use_t` Use Student-t (`n_eff - q` degrees of freedom) instead of normal inference.

`time` Time index for data frames: the name of a `Date`, `POSIXct` or numeric column of `data`, or a vector with one value per row. It is used to check that rows are sorted and have no gaps, and must have no missing values in the sample. Convert character dates first, e.g. with `as.Date()` or `zoo::as.yearmon()`.

`allow_gaps` Accept a time index with gaps, or a `subset` that skips rows (lags are then positional), with a warning.

`check_stationarity` Screen the response and the regressors with augmented Dickey-Fuller tests and warn when a unit root is not rejected.

`control` Advanced options, see `fgls_control()`.

`contrasts` Optional list of contrasts for factors, as in `stats::lm()`.

`model` Keep the model frame in the result.

`weights` Not supported: FGLS estimates its own weights. Use `hetero = "known"` with `variance` for known variances.

`...` Not used; unknown arguments are an error.

Value

An object of class `"fgls"`, a list with components including

`coefficients`, `vcov` Estimates and their covariance matrix.

`residuals`, `fitted.values` $y_t - x_t'\hat{\beta}$ and $x_t'\hat{\beta}$ on the full sample.

`innovations` Residuals of the final (filtered and/or weighted) regression.

`ar_order`, `rho`, `bic_path`, `nonexogeneity_share` Serial block (`NULL` without serial correction); more in `serial`.

`skedastic`, `weights` Heteroskedasticity block: selected covariates, coefficients, ψ, λ, degrees of freedom, fitted variances; `weights` are $1/\hat{v}_t$. For the Lasso, `skedastic$lambda` is the selected penalty on the scale of 'glmnet', whose objective is that of GP24 eq. (4) divided by the number of observations n of the skedastic regression, and `skedastic$lambda_paper` $= n\lambda$ is the same penalty on the scale of eq. (4). Both refer to the standardized candidate covariates.

`stages` Intermediate estimators (`stages$serial` is F-C). They are `"fgls"` objects with the same methods as the final fit; `vcov(fit$stages$serial, type =)` accepts `"robust"`, `"HC0"` and `"HC3"`.

`nobs`, `n_eff` Number of observations and of observations in the final regression ($T - k^*$).

Use the methods `summary()`, `coef()`, `vcov()`, `confint()`, `predict()`, `residuals()`, `plot()` and `compare_fgls()`.

Serial correlation (PG26)

The errors are approximated by an AR(k) process. Its coefficients are the coefficients on the lagged dependent variable in the Durbin regression of y_t on $y_{t-1}, \ldots, y_{t-k}$, $x_t, \ldots, x_{t-k}$ (eq. 14), and k is chosen by BIC over $0, \ldots, k_{max}$ on a common sample. The data are then quasi-differenced and the model is estimated by OLS on the filtered data (eq. 15), dropping the first k observations. Unlike Cochrane-Orcutt, this is consistent when the regressors are only *pre-determined* (uncorrelated with current and future innovations) rather than strictly exogenous. The default variance (`vcov =`

`"robust"`, eqs. 28–29) accounts for the estimation of the filter and is valid in both cases; `vcov = "naive"` (eq. 27) is valid only for exogenous regressors. The share by which the robust variance exceeds the naive one is reported as `nonexogeneity_share`. If the selected order is 0, FGLS equals OLS.

Heteroskedasticity (GP24)

The log of the squared OLS residuals (floored at `delta^2`) is regressed on candidate covariates z_t by an adaptive Lasso: a ridge first step gives weights $|\hat{\gamma}_j|^{-\psi}$, and the penalty λ and the exponent ψ are chosen jointly by 10-fold cross-validation with the same folds for every ψ. The exponentiated fit gives the weights of a weighted least-squares regression. Inference uses a sandwich variance with the bias correction of eq. (8), so it stays valid when the skedastic model is wrong. If the Lasso selects no covariate the weights are constant and FGLS equals OLS (no loss under homoskedasticity).

Both (Supplement S-4)

With `serial != "none"` and `hetero != "none"` the serial correction is applied first ("F-C"), the skedastic function is fitted to the residuals of the quasi-differenced regression with candidate covariates built from the *unfiltered* regressors, and the quasi-differenced regression is re-estimated by weighted least squares ("F-CH"). The F-C stage is kept in `fit$stages$serial`.

Choosing options

- Time series with possibly autocorrelated errors and no concern about heteroskedasticity: `hetero = "none"` (or `serial_fgls()`).
- Cross-sectional data: `serial = "none"` (or `lasso_fgls()`). Rows can be in any order.
- Lagged dependent variables among the regressors must be declared in `lagged_dep_vars`; the AR coefficients are then recovered by the Wallis ratio (PG26 Section 4.2).
- Errors known to be MA(1), e.g. multi-step forecast errors: `serial = "ma1"` (GMA estimator, PG26 Section 5).
- With several stochastic regressors the BIC that penalizes only the AR order can select `kmax`; use `control = fgls_control(ic_penalty = "params")`.
- Results involving cross-validation depend on the random folds: set `seed` for reproducibility.

References

Perron, P. and González-Coya, E. (2026). Feasible GLS for time series regression. *Journal of Econometrics*, 258, 106346. doi:10.1016/j.jeconom.2026.106346

González-Coya, E. and Perron, P. (2024). Estimation in the presence of heteroskedasticity of unknown form: a Lasso-based approach. *Journal of Econometric Methods*, 13(1), 29–48. doi:10.1515/jem20230007

Romano, J. P. and Wolf, M. (2017). Resurrecting weighted least squares. *Journal of Econometrics*, 197(1), 1–19. doi:10.1016/j.jeconom.2016.10.003

See Also

Other fgls: `compare_fgls()`, `fgls-methods`, `fgls_control()`, `lasso_fgls()`, `plot.fgls()`, `serial_fgls()`, `summary.fgls()`

Examples

```
# Serial correlation with a non-exogenous regressor (PG26 Table 1 design)
sim <- simulate_serial(300, seed = 1)
fit <- fgls(y ~ x, data = sim, hetero = "none")
summary(fit)

# Heteroskedasticity of unknown form (GP24)
het <- simulate_hetero(200, skedastic = "log2", seed = 2)
fit2 <- fgls(y ~ x, data = het, serial = "none", seed = 1)
fit2$skedastic$selected

# Both (Supplement S-4)
both <- simulate_combined(200, ar = 0.5, skedastic = "x2", seed = 3)
fit3 <- fgls(y ~ x, data = both, seed = 1)
compare_fgls(fit3$stages$serial, fit3)
```

---

fgls-broom *Tidy, glance and augment fgls fits*

Description

Methods for the 'broom' generics (from the 'generics' package), registered when 'generics' is loaded.

Usage

```
## S3 method for class 'fgls'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)

## S3 method for class 'fgls'
glance(x, ...)

## S3 method for class 'fgls'
augment(
  x,
  data = NULL,
  newdata = NULL,
  se_fit = FALSE,
  interval = c("none", "confidence"),
  conf.level = 0.95,
  ...
)
```

Arguments

x	An "fgls" object.
conf.int	Include confidence intervals.

conf.level	Confidence level.
...	Not used.
data	Data to augment: by default the model frame. It must have one row per observation used in the fit or, for a fit with na.action = na.exclude, one row per row of the original data (the excluded rows then get NA).
newdata	Optional new data; fitted values (and, if requested, standard errors and confidence limits) are added, but no residuals.
se_fit	Add the standard errors of the fitted means (.se.fit).
interval	"none" or "confidence": add confidence limits for the mean of the response (.lower, .upper) at level conf.level.

Value

A data frame (a tibble if 'tibble' is installed): one row per coefficient for tidy(), one row for glance(), one row per observation for augment() (columns .fitted, optionally .lower, .upper and .se.fit, then .resid, .innovation and, with a heteroskedasticity correction, .weight). nonexogeneity.share in glance() is NA when the share was not computed.

Examples

```
if (requireNamespace("broom", quietly = TRUE)) {
  fit <- serial_fgls(y ~ x, data = simulate_serial(200, seed = 1))
  broom::tidy(fit, conf.int = TRUE)
  broom::glance(fit)
  head(broom::augment(fit))
}
```

---

fgls-conditions *Conditions signaled by fgls*

Description

All warnings raised by the package inherit from class "fgls_warning" and all errors from "fgls_error", so they can be caught or muffled as a group with tryCatch(), withCallingHandlers() or suppressWarnings() (whose classes argument, available since R 4.0.0, filters by class).

Warning classes

- fgls_warning_nonstationary: the estimated autoregressive filter has a root on, inside or close to the unit circle.
- fgls_warning_collinearity: a matrix that should be positive definite was not numerically so (for example a nearly singular cross-product of the regressors, or a covariance matrix with a negative eigenvalue), and a pseudo-inverse or eigenvalue clipping was used instead. Collinear lag columns dropped from the Durbin regression (lags of the intercept, of a trend or of seasonal dummies) do not signal it: they are recorded in fit$serial$dropped_lags and, for serial-only fits, in the notes of summary.fgls().

- `fgls_warning_convergence`: an iterative routine did not fully converge.
- `fgls_warning_weight`: the estimated weights are extreme.
- `fgls_warning_sample_size`: the sample is small relative to the number of parameters, lags or folds.
- `fgls_warning_specification`: the specification looks unusual (e.g. no intercept, the selected order hits `kmax`).
- `fgls_warning_lagged_dependent`: a regressor looks like a lag of the dependent variable (also a specification warning).
- `fgls_warning_experimental`: the feature is experimental.

Error classes

`fgls_error_input` (invalid arguments) and its subclasses `fgls_error_missing`, `fgls_error_time_index`, `fgls_error_collinearity` and `fgls_error_insufficient_data`.

Examples

```
# an interior missing value in a time series is an error of class fgls_error_missing
df <- data.frame(y = c(1, 2, NA, 4, 5, 3, 2), x = c(2, 1, 3, 5, 4, 2, 1))
tryCatch(serial_fgls(y ~ x, data = df), fgls_error = function(e) class(e)[1])

# warnings can be caught by class too
withCallingHandlers(
  serial_fgls(y ~ x - 1, data = simulate_serial(100, seed = 1)),
  fgls_warning_specification = function(w) invokeRestart("muffleWarning")
)$ar_order
```

---

fgls-methods *Methods for fgls fits*

Description

Standard accessors for objects returned by `fgls()`, `serial_fgls()`, `lasso_fgls()` and the benchmark estimators. They make the fits work with generic tools such as `lmtest::coeftest()` and `lmtest::coefci()`.

Usage

```
## S3 method for class 'fgls'
vcov(object, type = NULL, ...)

## S3 method for class 'fgls'
confint(object, parm, level = 0.95, type = NULL, ...)

## S3 method for class 'fgls'
residuals(object, type = c("response", "innovations"), ...)
```

```
## S3 method for class 'fgls'
fitted(object, ...)

## S3 method for class 'fgls'
predict(
  object,
  newdata,
  se.fit = FALSE,
  interval = c("none", "confidence"),
  level = 0.95,
  type = "response",
  na.action = stats::na.pass,
  ...
)

## S3 method for class 'fgls'
nobs(object, ...)

## S3 method for class 'fgls'
df.residual(object, ...)

## S3 method for class 'fgls'
formula(x, ...)

## S3 method for class 'fgls'
model.matrix(object, ...)

## S3 method for class 'fgls'
model.frame(formula, ...)

## S3 method for class 'fgls'
terms(x, ...)

## S3 method for class 'fgls'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

`object`, `x`, `formula`	An `"fgls"` object.
`type`	For `vcov()` and `confint()`: `NULL` (the variance chosen when fitting) or another variance type to compute from the stored estimates: `"robust"`, `"naive"`, `"nonrobust"`, `"HC0"`–`"HC3"` (where available) or `"bootstrap"` (re-runs the estimator); see the section *Variance types*. For `residuals()`: `"response"` ($y_t - x_t'\hat{\beta}$, full sample) or `"innovations"` (residuals of the final filtered and/or weighted regression). For `predict()`: only `"response"` (the mean of the response) is supported.
`...`	Not used.

parm	Coefficients (names or positions); all by default.
level	Confidence level.
newdata	Optional data frame in which to look for the regressors. Without it, predictions are for the observations of the fit, padded with NA for the rows dropped by stats::na.exclude().
se.fit	Return standard errors of the predicted means.
interval	"none" or "confidence" (for the mean of the response).
na.action	Missing-value handling for newdata.
digits	Number of significant digits to print.

Details

Inference is asymptotically normal (as in the papers) unless the fit was made with use_t = TRUE. Accordingly, df.residual() returns Inf for normal inference, so that lmtest::coeftest() reproduces summary.fgls(), and $n_{eff} - q$ otherwise. nobs() returns the number of observations in the final regression, $n_{eff} = T - k^*$.

Value

The usual values: a coefficient covariance matrix (vcov), a matrix of confidence limits (confint), vectors of residuals or fitted values, predictions (a vector, or a list/matrix with se.fit/interval), or the corresponding model components. The print() method returns x invisibly.

Variances and the sandwich package

The variance estimators of the papers are available through vcov(fit, type = ...) and can be passed to other tools (for example lmtest::coeftest(fit, vcov. = vcov(fit, type = "naive"))). The package provides no sandwich::estfun() or sandwich::bread() methods, because the robust variance of PG26 eq. (29) and the bias-corrected sandwich of GP24 eq. (8) are not of the generic form those functions assume: sandwich::vcovHAC() and related functions would *not* reproduce them.

Variance types

vcov(fit, type = t) returns the stored covariance when t is the type the fit was made with (e.g. "robust", or "HAC" for ols_robust(), whose stored vcov_type label also shows the kernel and bandwidth). Other types are recomputed from the stored estimates only for fits made by fgls(), serial_fgls() and lasso_fgls(), and for the F-C stage of a combined fit (fit$stages$serial, types "robust", "HC0" and "HC3"). The benchmark estimators store only the covariance they were fitted with; refit them with another vcov instead.

See Also

Other fgls: compare_fgls(), fgls(), fgls_control(), lasso_fgls(), plot.fgls(), serial_fgls(), summary.fgls()

Examples

```
sim <- simulate_serial(200, seed = 1)
fit <- serial_fgls(y ~ x, data = sim)
vcov(fit)
vcov(fit, type = "naive")
confint(fit)
predict(fit, newdata = data.frame(x = c(-1, 0, 1)), interval = "confidence")
head(residuals(fit, type = "innovations"))
nobs(fit)
```

`fgls_control` *Advanced options for fgls*

Description

Tuning options of `fgls()` that most users never change. The defaults follow the papers and the Python package 'pyfgls'.

Usage

```
fgls_control(
  ic_penalty = c("order", "params"),
  min_lags = NULL,
  fold_scheme = c("random", "blocked"),
  foldid = NULL,
  cv_weights = c("fold", "full"),
  ridge_scale = c("standardized", "original"),
  n_lambda = 100L,
  lambda_min_ratio = NULL,
  lasso_engine = c("lars", "glmnet"),
  glmnet_thresh = 1e-07,
  delta_scale = FALSE,
  transforms = c("linear", "poly2", "log", "trig"),
  svr_cost = c(0.1, 1, 10, 100),
  svr_epsilon = c(0.05, 0.25, 0.5, 1),
  n_boot = 999L,
  bootstrap = c("wild", "iid"),
  wild_dist = c("rademacher", "mammen"),
  boot_seed = NULL,
  cores = 1L
)
```

Arguments

`ic_penalty`	Penalty of the lag-order criterion: `"order"` (default, PG26 eq. 14) penalizes the AR order; `"params"` penalizes every coefficient a lag adds to the Durbin regression, which avoids over-selection with many stochastic regressors.

`min_lags` Lower bound on the selected AR order: `NULL` (default), a non-negative integer, or `"adjusted"` for $\lceil (\ln T)^{3/2} \rceil$ (PG26 Supplement, Lemma A.2).

`fold_scheme` Cross-validation folds of the skedastic fit: `"random"` (default) or `"blocked"` (contiguous blocks).

`foldid` Optional integer vector of fold labels (one per observation entering the skedastic fit), overriding `fold_scheme` and the seed.

`cv_weights` `"fold"` (default) re-estimates the ridge first step, and hence the adaptive weights, on every training fold; `"full"` computes the weights once on the full sample (the common practice with `glmnet::cv.glmnet()` and `penalty.factor`).

`ridge_scale` Scale on which the ridge coefficients define the adaptive weights: `"standardized"` (default) or `"original"` units of `z`.

`n_lambda` Length of the Lasso penalty grid.

`lambda_min_ratio` Smallest penalty relative to the largest; default `1e-4` when there are more observations than candidate covariates and `1e-2` otherwise (as in 'glmnet').

`lasso_engine` Solver of the Lasso problems: `"lars"` (default) computes the exact solution path by the LARS homotopy with the Lasso modification, as 'pyfgls' does; `"glmnet"` uses `glmnet::glmnet()` (coordinate descent) followed by an exact active-set refinement. Both give the same solutions; `"lars"` is faster for the moderate numbers of candidate covariates typical here.

`glmnet_thresh` Convergence threshold passed to `glmnet::glmnet()` when `lasso_engine = "glmnet"`.

`delta_scale` Use the scale-aware floor `delta * sigma_OLS` in $\log \max(\hat{u}_t^2, \delta^2)$.

`transforms` Transform families used by `z = "auto"`; see `make_dictionary()`.

`svr_cost, svr_epsilon` Grids for the SVR skedastic fit, multiplied by the standard deviation of the response.

`n_boot` Number of bootstrap replications for `vcov = "bootstrap"`.

`bootstrap` Innovation scheme of the sieve bootstrap: `"wild"` (default) or `"iid"` resampling.

`wild_dist` Wild-bootstrap multipliers: `"rademacher"` (default) or `"mammen"`.

`boot_seed` Seed of the bootstrap; default the `seed` of the fit.

`cores` Number of cores for the bootstrap (forked processes via 'parallel'; ignored on Windows). Defaults to 1.

Value

A list of class `"fgls_control"`.

See Also

Other fgls: `compare_fgls()`, `fgls()`, `fgls-methods`, `lasso_fgls()`, `plot.fgls()`, `serial_fgls()`, `summary.fgls()`

Examples

```
fgls_control(ic_penalty = "params", n_boot = 199)
```

`lasso_fgls` *Feasible GLS under heteroskedasticity of unknown form (González-Coya and Perron, 2024)*

Description

Shortcut for `fgls()` with `serial = "none"`. The log of the squared OLS residuals is regressed on candidate covariates `z` by an adaptive Lasso tuned by cross-validation; the exponentiated fit gives the weights of a weighted least-squares regression. Inference uses the sandwich of eq. (6) with the bias correction of eq. (8), so it remains valid when the skedastic model is misspecified.

Usage

```
lasso_fgls(
  formula,
  data,
  subset,
  na.action = stats::na.fail,
  hetero = c("lasso", "svr", "wls_s1", "wls_s2", "known", "none"),
  vcov = c("robust", "nonrobust", "HC0", "HC1", "HC2", "HC3", "bootstrap"),
  z = "auto",
  seed = NULL,
  ...
)
```

Arguments

`formula`	A model formula, as in `stats::lm()`. Include an intercept unless the model truly has none.
`data`	A data frame, a list, a `ts`/`mts` object, or a 'zoo'/'xts' object containing the variables in the formula. Rows must be in time order (unless `serial = "none"`).
`subset`	Optional expression selecting rows, evaluated in `data`. For time series without a time index (`time`, or a ts/zoo `data`) the selected rows must be an increasing block of consecutive rows (e.g. `subset = 51:200`): repeated or reversed rows are an error, and so are skipped rows unless `allow_gaps = TRUE`.
`na.action`	`stats::na.fail()` (default) stops on missing values; `stats::na.omit()` or `stats::na.exclude()` drop them. For time series only leading and trailing missing values can be dropped. With `na.exclude`, `residuals()` and `fitted()` are padded with `NA` to the rows selected by `subset`, as for `stats::lm()`.
`hetero`	Skedastic-function estimator: `"lasso"` (default), `"svr"`, `"wls_s1"`, `"wls_s2"`, `"known"` or `"none"` (OLS with HC3 standard errors).
`vcov`	`"robust"` (default; eq. 8 for Lasso/SVR, eq. 7 for WLS-S1/S2, HC3 otherwise), `"nonrobust"`, `"HC0"`–`"HC3"` or `"bootstrap"` (wild bootstrap).

z	Candidate covariates of the skedastic function: "auto" (default) builds $\{x, x^2, \log x, (\log x)^2, \cos x, \cos 2$ from every non-constant regressor (see make_dictionary()); a character vector of regressor names; a one-sided formula evaluated in data (e.g. ~ x + I(x^2) + w); or a numeric matrix / data frame with one row per row of data. With hetero = "wls_s1" all candidates must be strictly positive, so give them explicitly, e.g. z = "x" or z = ~ x + I(x^2) for a positive regressor x.
seed	Optional seed for the cross-validation folds and the bootstrap. The session's random number generator state is restored afterwards.
...	Further arguments passed to fgls(), e.g. variance, cv_folds, lambda_rule, weight_clip, use_t, control.

Value

An object of class "fgls"; see fgls().

References

Perron, P. and González-Coya, E. (2026). Feasible GLS for time series regression. *Journal of Econometrics*, 258, 106346. doi:10.1016/j.jeconom.2026.106346

González-Coya, E. and Perron, P. (2024). Estimation in the presence of heteroskedasticity of unknown form: a Lasso-based approach. *Journal of Econometric Methods*, 13(1), 29–48. doi:10.1515/jem20230007

Romano, J. P. and Wolf, M. (2017). Resurrecting weighted least squares. *Journal of Econometrics*, 197(1), 1–19. doi:10.1016/j.jeconom.2016.10.003

See Also

Other fgls: compare_fgls(), fgls(), fgls-methods, fgls_control(), plot.fgls(), serial_fgls(), summary.fgls()

Other heteroskedasticity: make_dictionary()

Examples

```
het <- simulate_hetero(200, skedastic = "x2", seed = 1)
fit <- lasso_fgls(y ~ x, data = het, seed = 1)
fit
fit$skedastic$selected
ols <- lasso_fgls(y ~ x, data = het, hetero = "none")
compare_fgls(OLS = ols, FGLS = fit)
```

`make_dictionary` *Candidate covariates for the skedastic function*

Description

Builds the default dictionary of candidate covariates used by `fgls()` and `lasso_fgls()` with `z = "auto"`: for every non-constant regressor x, the columns x, x^2, $\log x$ and $(\log x)^2$ (only when $x > 0$), $\cos x$ and $\cos 2x$. This contains the specification $z = (1, x, [\log x]^2, x^2, \cos x, \cos 2x)$ of the simulations of González-Coya and Perron (2024); the intercept of the skedastic regression is always included and never penalized. Binary (dummy) columns are passed through untransformed, undefined or constant transforms are skipped and duplicated columns are removed.

Usage

```
make_dictionary(
  x,
  transforms = c("linear", "poly2", "log", "trig"),
  skip = NULL
)
```

Arguments

`x`	A numeric matrix or data frame of regressors (constant columns are ignored), or a numeric vector.
`transforms`	Transform families: `"linear"` (x), `"poly2"` (x^2), `"log"` ($\log x$, $(\log x)^2$ for positive x) and `"trig"` ($\cos x$, $\cos 2x$).
`skip`	Columns (names or positions) to leave out, e.g. lagged dependent variables.

Value

A numeric matrix with one column per candidate covariate and informative column names.

References

Perron, P. and González-Coya, E. (2026). Feasible GLS for time series regression. *Journal of Econometrics*, 258, 106346. doi:10.1016/j.jeconom.2026.106346

González-Coya, E. and Perron, P. (2024). Estimation in the presence of heteroskedasticity of unknown form: a Lasso-based approach. *Journal of Econometric Methods*, 13(1), 29–48. doi:10.1515/jem20230007

Romano, J. P. and Wolf, M. (2017). Resurrecting weighted least squares. *Journal of Econometrics*, 197(1), 1–19. doi:10.1016/j.jeconom.2016.10.003

See Also

Other heteroskedasticity: `lasso_fgls()`

Examples

```
X <- cbind(const = 1, x = seq(1, 4, length.out = 10), d = rep(0:1, 5))
colnames(make_dictionary(X))
colnames(make_dictionary(X, transforms = c("linear", "poly2")))
```

`ols_robust` *Benchmark estimators*

Description

Estimators used as comparisons in the papers and in `compare_fgls()`:

Usage

```
ols_robust(
  formula,
  data,
  subset,
  na.action = stats::na.fail,
  vcov = c("HAC", "NW", "HC0", "HC1", "HC2", "HC3", "nonrobust"),
  kernel = NULL,
  bandwidth = NULL,
  small_sample = FALSE,
  use_t = FALSE,
  time = NULL,
  allow_gaps = FALSE,
  model = TRUE
)

cochrane_orcutt(
  formula,
  data,
  subset,
  na.action = stats::na.fail,
  ar_order = 1L,
  iterate = FALSE,
  max_iter = 100L,
  tol = 1e-08,
  use_t = FALSE,
  time = NULL,
  allow_gaps = FALSE,
  model = TRUE
)

durbin_estimator(
  formula,
  data,
```

```
  subset,
  na.action = stats::na.fail,
  ar_order = NULL,
  kmax = NULL,
  use_t = FALSE,
  time = NULL,
  allow_gaps = FALSE,
  model = TRUE
)

infeasible_gls(
  formula,
  data,
  subset,
  na.action = stats::na.fail,
  rho = NULL,
  omega = NULL,
  use_t = FALSE,
  time = NULL,
  allow_gaps = FALSE,
  model = TRUE
)

hybrid_gls(
  formula,
  data,
  subset,
  na.action = stats::na.fail,
  alpha = 0.05,
  lags = 1L,
  seed = NULL,
  use_t = FALSE,
  time = NULL,
  allow_gaps = FALSE,
  model = TRUE
)
```

Arguments

formula	A model formula, as in stats::lm(). Include an intercept unless the model truly has none.
data	A data frame, a list, a ts/mts object, or a 'zoo'/'xts' object containing the variables in the formula. Rows must be in time order (unless serial = "none").
subset	Optional expression selecting rows, evaluated in data. For time series without a time index (time, or a ts/zoo data) the selected rows must be an increasing block of consecutive rows (e.g. subset = 51:200): repeated or reversed rows are an error, and so are skipped rows unless allow_gaps = TRUE.
na.action	stats::na.fail() (default) stops on missing values; stats::na.omit() or

	`stats::na.exclude()` drop them. For time series only leading and trailing missing values can be dropped. With `na.exclude`, `residuals()` and `fitted()` are padded with `NA` to the rows selected by `subset`, as for `stats::lm()`.
`vcov`	Covariance estimator for `ols_robust()`.
`kernel`, `bandwidth`	Optional overrides for the HAC estimator: kernel `"qs"` or `"bartlett"`, bandwidth `"andrews"`, `"nw"` or a positive number.
`small_sample`	Multiply HAC covariances by $T/(T-q)$.
`use_t`	Use Student-t (`n_eff - q` degrees of freedom) instead of normal inference.
`time`	Time index for data frames: the name of a `Date`, `POSIXct` or numeric column of `data`, or a vector with one value per row. It is used to check that rows are sorted and have no gaps, and must have no missing values in the sample. Convert character dates first, e.g. with `as.Date()` or `zoo::as.yearmon()`.
`allow_gaps`	Accept a time index with gaps, or a `subset` that skips rows (lags are then positional), with a warning.
`model`	Keep the model frame in the result.
`ar_order`	AR order (`cochrane_orcutt()`, `durbin_estimator()`).
`iterate`	Iterate Cochrane-Orcutt to convergence.
`max_iter`, `tol`	Iteration limit and convergence tolerance.
`kmax`	Largest AR order considered: `NULL` (default, $\lfloor 12(T/200)^{1/4} \rfloor$), an integer, or `"theory"` ($\lfloor (T/\ln T)^{1/3} \rfloor$). It is capped so that the Durbin regression uses at most half of the sample as regressors.
`rho`, `omega`	Known AR coefficients or known error covariance matrix.
`alpha`	Level of the Breusch-Godfrey test (`hybrid_gls()`), strictly between 0 and 1.
`lags`	Number of lags of the Breusch-Godfrey test, a positive integer.
`seed`	Optional seed for the random AR(1) coefficient that `hybrid_gls()` draws when the Breusch-Godfrey test rejects. The session's random number generator state is restored afterwards.

Details

- `ols_robust()`: ordinary least squares with heteroskedasticity and autocorrelation consistent standard errors (`vcov = "HAC"`: Andrews' 1991 quadratic-spectral kernel with automatic bandwidth, the benchmark of PG26; `"NW"`: Newey-West Bartlett kernel with $\lfloor 4(T/100)^{2/9} \rfloor$ lags) or White-type standard errors (`"HC0"`–`"HC3"`).
- `cochrane_orcutt()`: two-step (or iterated) Cochrane-Orcutt FGLS. The two-step version estimates the AR coefficients from OLS residuals and is **not valid with non-exogenous regressors** (PG26 Section 3); use `serial_fgls()` instead.
- `durbin_estimator()`: the Durbin regression (PG26 eq. 14) of y_t on y_{t-j}, x_t and x_{t-j}, $j = 1, \ldots, k$, with the AR order k chosen by BIC unless given. The coefficients on the stochastic regressors are those on x_t. Lags of the constant, of trends and of periodic dummies are collinear with the regressors and dropped, so the Durbin coefficients c on these regressors estimate $(I - \sum_j \rho_j P_j)\beta$, where the dropped lags are $x_{t-j} = x_t P_j$ (plus kept lags). They

are converted back to $\hat{\beta} = (I - \sum_j \hat{\rho}_j P_j)^{-1} c$, e.g. $c/(1 - \sum_j \hat{\rho}_j)$ for the intercept, with a delta-method covariance, and are NA (with a note) when that is not possible. Residuals, fitted values and predictions use these coefficients; `innovations` are the residuals of the Durbin regression. (The Python package 'pyfgls' reports the untransformed c.)

- `infeasible_gls()`: GLS with known AR coefficients (`rho`) or a known error covariance matrix (`omega`, up to scale); a simulation benchmark.
- `hybrid_gls()`: the hybrid OLS/GLS estimator of PG26 Supplement S-1 (exogenous regressors only): OLS when the Breusch-Godfrey test does not reject, otherwise quasi-differencing with a random AR(1) coefficient of the sign of the first-order residual autocorrelation.

Value

An object of class `"fgls"` (see `fgls()`). The non-exogeneity share of PG26 eq. (29) is not computed for these estimators and is NA.

References

Perron, P. and González-Coya, E. (2026). Feasible GLS for time series regression. *Journal of Econometrics*, 258, 106346. doi:10.1016/j.jeconom.2026.106346

González-Coya, E. and Perron, P. (2024). Estimation in the presence of heteroskedasticity of unknown form: a Lasso-based approach. *Journal of Econometric Methods*, 13(1), 29–48. doi:10.1515/jem20230007

Romano, J. P. and Wolf, M. (2017). Resurrecting weighted least squares. *Journal of Econometrics*, 197(1), 1–19. doi:10.1016/j.jeconom.2016.10.003

Examples

```
sim <- simulate_serial(300, seed = 1)
ols_robust(y ~ x, data = sim)
cochrane_orcutt(y ~ x, data = sim)
durbin_estimator(y ~ x, data = sim)
infeasible_gls(y ~ x, data = sim, rho = 0.8)
```

---

plot.fgls *Diagnostic plots for fgls fits*

Description

Base-graphics diagnostics: (1) residuals against fitted values; (2) the autocorrelation function of the innovations (the residuals of the final filtered and/or weighted regression, which should look like white noise); (3) the lag-order criterion path with the selected order (serial correction); (4) the fitted variances against the fitted values (heteroskedasticity correction), or a normal quantile plot of the innovations otherwise.

Usage

```
## S3 method for class 'fgls'
plot(x, which = 1:4, ask = FALSE, lag.max = 20L, ...)
```

Arguments

x	An "fgls" object.
which	Panels to draw (subset of 1:4).
ask	Ask before each new page.
lag.max	Number of lags of the autocorrelation function.
...	Further graphical arguments passed to plot() (or stats::qqnorm() for panel 4). main, xlab and ylab replace the default titles and axis labels of every panel drawn.

Value

x, invisibly.

See Also

Other fgls: compare_fgls(), fgls(), fgls-methods, fgls_control(), lasso_fgls(), serial_fgls(), summary.fgls()

Examples

```
fit <- fgls(y ~ x, data = simulate_combined(200, seed = 1), seed = 1)
plot(fit)
```

---

serial_fgls *Feasible GLS for serially correlated errors (Perron and González-Coya, 2026)*

Description

Shortcut for fgls() with hetero = "none": the errors are approximated by an AR(k) process whose coefficients come from the Durbin regression and whose order is chosen by BIC (eq. 14), the data are quasi-differenced and OLS is applied (eq. 15). The default variance is the exogeneity-robust estimator of eqs. (28)–(29), valid when the regressors are only pre-determined.

Usage

```
serial_fgls(
  formula,
  data,
  subset,
  na.action = stats::na.fail,
  serial = c("auto", "fixed", "ma1"),
  vcov = c("robust", "naive", "HC0", "HC1", "HC2", "HC3", "bootstrap"),
  ar_order = NULL,
  kmax = NULL,
  ic = c("bic", "aic"),
```

```
  lagged_dep_vars = NULL,
  exog = c("predetermined", "exogenous"),
  ...
)
```

Arguments

formula	A model formula, as in stats::lm(). Include an intercept unless the model truly has none.
data	A data frame, a list, a ts/mts object, or a 'zoo'/'xts' object containing the variables in the formula. Rows must be in time order (unless serial = "none").
subset	Optional expression selecting rows, evaluated in data. For time series without a time index (time, or a ts/zoo data) the selected rows must be an increasing block of consecutive rows (e.g. subset = 51:200): repeated or reversed rows are an error, and so are skipped rows unless allow_gaps = TRUE.
na.action	stats::na.fail() (default) stops on missing values; stats::na.omit() or stats::na.exclude() drop them. For time series only leading and trailing missing values can be dropped. With na.exclude, residuals() and fitted() are padded with NA to the rows selected by subset, as for stats::lm().
serial	"auto" (default), "fixed" or "ma1" (GMA estimator for MA(1) errors, PG26 Section 5; its only variance is vcov = "robust", the GMA covariance $s^2(X^{*\prime}X^{*})^{-1}$).
vcov	"robust" (default, eq. 29), "naive" (eq. 27, exogenous regressors only), "HC0"–"HC3" (eq. 29 with heteroskedasticity weights) or "bootstrap" (sieve bootstrap).
ar_order	Fixed AR order (implies serial = "fixed").
kmax	Largest AR order considered: NULL (default, $\lfloor 12(T/200)^{1/4} \rfloor$), an integer, or "theory" ($\lfloor (T/\ln T)^{1/3} \rfloor$). It is capped so that the Durbin regression uses at most half of the sample as regressors.
ic	Lag-order criterion: "bic" (default) or "aic".
lagged_dep_vars	Names (or positions) of model-matrix columns that are lags of the dependent variable (not available with serial = "ma1").
exog	Assumption on the regressors: "predetermined" (default) or "exogenous". Estimates do not depend on it; it sets the default of first_stage_correction and the notes of the summary.
...	Further arguments passed to fgls(), e.g. wallis_regressor, use_t, time, control.

Value

An object of class "fgls"; see fgls().

References

Perron, P. and González-Coya, E. (2026). Feasible GLS for time series regression. *Journal of Econometrics*, 258, 106346. doi:10.1016/j.jeconom.2026.106346

González-Coya, E. and Perron, P. (2024). Estimation in the presence of heteroskedasticity of unknown form: a Lasso-based approach. *Journal of Econometric Methods*, 13(1), 29–48. doi:10.1515/jem20230007

Romano, J. P. and Wolf, M. (2017). Resurrecting weighted least squares. *Journal of Econometrics*, 197(1), 1–19. doi:10.1016/j.jeconom.2016.10.003

See Also

Other fgls: `compare_fgls()`, `fgls()`, `fgls-methods`, `fgls_control()`, `lasso_fgls()`, `plot.fgls()`, `summary.fgls()`

Examples

```
sim <- simulate_serial(500, seed = 42)
fit <- serial_fgls(y ~ x, data = sim)
fit
fit$ar_order
fit$nonexogeneity_share

# Lagged dependent variable among the regressors (Wallis recovery)
df <- data.frame(y = sim$y[-1], ylag = sim$y[-500], x = sim$x[-1])
serial_fgls(y ~ ylag + x, data = df, lagged_dep_vars = "ylag")
```

---

simulate_serial *Simulate the designs of the papers*

Description

Data-generating processes used in the Monte Carlo studies of Perron and González-Coya (2026) (`simulate_serial()`, `simulate_combined()`, `simulate_predictive()`) and González-Coya and Perron (2024) (`simulate_hetero()`). They are handy for examples, teaching and replication.

Usage

```
simulate_serial(
  n,
  ar = 0.8,
  ma = numeric(0),
  rho_x = 0,
  gamma = 1,
  beta = c(1, 1),
  burn = 0,
  seed = NULL
)

simulate_hetero(
  n,
  skedastic = "x2",
```

```
  beta = c(1, 1),
  n_extra = 0L,
  extra_corr = 0,
  seed = NULL
)

simulate_combined(
  n,
  ar = 0.5,
  ma = numeric(0),
  skedastic = "x2",
  beta = c(1, 1),
  driver = c("x", "mixture"),
  phi_prob = 0.75,
  gamma = 0,
  burn = 100L,
  seed = NULL
)

simulate_predictive(
  n,
  theta = 0.5,
  rho_x = 0,
  gamma = 0,
  beta = c(1, 1),
  burn = 100L,
  seed = NULL
)
```

Arguments

n	Number of observations returned.
ar, ma	Autoregressive and moving-average coefficients of the errors.
rho_x	Autoregressive coefficient of the regressor.
gamma	Feedback from past innovations to the regressor; the regressor is not exogenous when `gamma != 0`.
beta	Intercept and slope.
burn	Number of initial observations discarded.
seed	Optional seed. The random number generator state of the session is restored afterwards.
skedastic	Skedastic function: one of `"homoskedastic"`, `"x"`, `"x2"`, `"log2"` ($[\log x]^2$), `"exp"` ($\exp(0.2x + 0.2x^2)$), `"step"`, or a function of `x`.
n_extra	Number of extra candidate covariates `w1, w2, ...` (columns of the result, not regressors).
extra_corr	Weight c of `x` in the extra covariates.
driver	Variance driver of `simulate_combined()`: `"x"` or `"mixture"`.

`phi_prob`	Probability that the variance driver equals `x` when `driver = "mixture"`.
`theta`	MA(1) coefficient of `simulate_predictive()`.

Details

- `simulate_serial()`: $y_t = \beta_0 + \beta_1 x_t + u_t$ with ARMA errors driven by standard normal innovations e_t and a regressor $x_t = \rho_x x_{t-1} + v_t + \gamma e_{t-1}$, which is pre-determined and exogenous only when $\gamma = 0$. The defaults ($\rho_x = 0$, $\gamma = 1$, AR(1) coefficient 0.8, no burn-in, $u_0 = 0$) give the design of Table 1 of the 2026 paper.
- `simulate_hetero()`: $y_t = \alpha + \beta x_t + \sqrt{v(x_t)}\varepsilon_t$, $x_t \sim U[1, 4]$ (eq. 9 of the 2024 paper), optionally with extra candidate covariates $w_j = cx + (1 - c)q_j$, $q_j \sim U[1, 4]$.
- `simulate_combined()`: ARMA errors whose innovations are heteroskedastic, $e_t = \sqrt{v(w_t)}\varepsilon_t$, with $w_t = x_t$ (`driver = "x"`) or $w_t = \phi_t x_t + (1 - \phi_t)q_t$, $\phi_t \sim$ Bernoulli(`phi_prob`) (`driver = "mixture"`) (Supplement S-4).
- `simulate_predictive()`: the predictive regression $y_{t+2} = \alpha + \beta x_t + u_{t+2}$ with MA(1) errors (Supplement S-3), returned aligned so that row t holds y_{t+2} and x_t.

Value

A data frame with columns `y` and `x` (and `w` or `w1, w2, ...` where relevant), with attributes `beta` (true coefficients), `u` (errors), `e` (innovations) and `info` (design parameters and, for the heteroskedastic designs, the true variances `v`).

References

Perron, P. and González-Coya, E. (2026). Feasible GLS for time series regression. *Journal of Econometrics*, 258, 106346. doi:10.1016/j.jeconom.2026.106346

González-Coya, E. and Perron, P. (2024). Estimation in the presence of heteroskedasticity of unknown form: a Lasso-based approach. *Journal of Econometric Methods*, 13(1), 29–48. doi:10.1515/jem20230007

Examples

```
sim <- simulate_serial(200, seed = 1)
head(sim)
attr(sim, "beta")

het <- simulate_hetero(100, skedastic = "log2", seed = 2)
summary(attr(het, "info")$v)
```

summary.fgls *Summarize an fgls fit*

Description

The summary contains the coefficient table (estimates, standard errors, z or t statistics, p-values and confidence intervals) and the main by-products of the estimator: the selected AR order k^* and AR coefficients with the non-exogeneity share for serial correlation, and the selected candidate covariates, the tuning parameters and the degrees of freedom of the skedastic fit for heteroskedasticity.

Usage

```
## S3 method for class 'fgls'
summary(object, level = 0.95, ...)

## S3 method for class 'summary.fgls'
print(
  x,
  digits = max(3L, getOption("digits") - 3L),
  signif.stars = getOption("show.signif.stars"),
  ...
)
```

Arguments

object	An "fgls" object.
level	Confidence level of the intervals.
...	Not used.
x	A "summary.fgls" object.
digits	Number of significant digits.
signif.stars	Print significance stars.

Value

summary() returns an object of class "summary.fgls": a list with the call, the coefficient table (coefficients), the confidence intervals (conf.int), a named list of header items (info) and the notes. Its print() method returns its argument invisibly.

See Also

Other fgls: compare_fgls(), fgls(), fgls-methods, fgls_control(), lasso_fgls(), plot.fgls(), serial_fgls()

Examples

```
sim <- simulate_serial(200, seed = 1)
s <- summary(serial_fgls(y ~ x, data = sim))
s
s$coefficients
```

`test_predetermined` *Test whether lags of the regressors are missing (pre-determinedness)*

Description

Experimental. A variable-addition LM test in the spirit of Remark 9 of Perron and González-Coya (2026): FGLS needs the regressors to be pre-determined (uncorrelated with current and future innovations), which can fail when lags of the regressors belong in the model. The steps are (a) the Durbin regression (AR order by BIC) giving $\hat{\beta}^D$; (b) the residuals $\hat{u}^D = y - X\hat{\beta}^D$, demeaned; (c) for $l = 1, \ldots,$ `max_lag`, the statistic $LM_l = n(SSR_0 - SSR_1)/SSR_0$, asymptotically χ^2 with as many degrees of freedom as added lags, from the auxiliary regressions without and with lags $1..l$ of the stochastic regressors.

Usage

```
test_predetermined(
  formula,
  data,
  max_lag = 4L,
  ar_order = NULL,
  kmax = NULL,
  regressors = NULL,
  alpha = 0.05,
  scale = c("innovations", "errors"),
  na.action = stats::na.fail,
  time = NULL,
  allow_gaps = FALSE
)

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

Arguments

`formula`	A model formula, as in `stats::lm()`. Include an intercept unless the model truly has none.
`data`	A data frame, a list, a `ts`/`mts` object, or a 'zoo'/'xts' object containing the variables in the formula. Rows must be in time order.
`max_lag`	Largest number of lags added.

ar_order	Fixed AR order of the Durbin regression; NULL (default) selects it by BIC up to kmax.
kmax	Largest AR order considered: NULL (default, $\lfloor 12(T/200)^{1/4} \rfloor$), an integer, or "theory" ($\lfloor (T/\ln T)^{1/3} \rfloor$). It is capped so that the Durbin regression uses at most half of the sample as regressors.
regressors	Names of the regressors whose lags are tested (default: all non-constant regressors).
alpha	Level used for the advice.
scale	"innovations" (default) or "errors".
na.action	stats::na.fail() (default) stops on missing values; stats::na.omit() or stats::na.exclude() drop them. For time series only leading and trailing missing values can be dropped. With na.exclude, residuals() and fitted() are padded with NA to the rows selected by subset, as for stats::lm().
time	Time index for data frames: the name of a Date, POSIXct or numeric column of data, or a vector with one value per row. It is used to check that rows are sorted and have no gaps, and must have no missing values in the sample. Convert character dates first, e.g. with as.Date() or zoo::as.yearmon().
allow_gaps	Accept a time index with gaps, or a subset that skips rows (lags are then positional), with a warning.
x	A "fgls_predetermined" object.
...	Not used.

Details

With scale = "innovations" (default) the auxiliary regression is run on the innovation scale: $\hat{u}^D$ and all regressors are filtered with the Durbin AR filter and lagged $\hat{u}^D$ enter as nuisance regressors, which keeps the χ^2 approximation valid with pre-determined but non-exogenous regressors. scale = "errors" uses $\hat{u}^D$ and the raw regressors (the literal reading of Remark 9).

Value

An object of class "fgls_predetermined": a list with

- table: a data frame with columns lags, LM, df and p.value, one row per number of added lags;
- ar_order: the AR order of the Durbin regression;
- regressors: the names of the regressors whose lags were tested;
- alpha: the level used for the advice;
- rejected: TRUE if some test rejects at level alpha;
- advice: a sentence summarizing the result;
- scale: the scale used, "innovations" or "errors".

The print method returns x invisibly.

References

Perron, P. and González-Coya, E. (2026). Feasible GLS for time series regression. *Journal of Econometrics*, 258, 106346. doi:10.1016/j.jeconom.2026.106346

González-Coya, E. and Perron, P. (2024). Estimation in the presence of heteroskedasticity of unknown form: a Lasso-based approach. *Journal of Econometric Methods*, 13(1), 29–48. doi:10.1515/jem20230007

Romano, J. P. and Wolf, M. (2017). Resurrecting weighted least squares. *Journal of Econometrics*, 197(1), 1–19. doi:10.1016/j.jeconom.2016.10.003

See Also

Other diagnostics: `adf_test()`, `weight_diagnostics()`

Examples

```
sim <- simulate_serial(300, seed = 1)
test_predetermined(y ~ x, data = sim)
```

`weight_diagnostics` *Diagnostics of an fgls fit*

Description

- `weight_diagnostics()`: how concentrated the FGLS weights $1/\hat{v}_t$ are: the ratio of the largest to the smallest fitted variance, the Kish effective sample size $(\sum w)^2/\sum w^2$ as a share of the sample, and the share of the total weight carried by the top 1% of observations.
- `ar_roots()`: roots of the estimated lag polynomial $1-\hat{\rho}_1 z-\cdots-\hat{\rho}_k z^k$; the filter is invertible when all lie outside the unit circle.
- `nonexogeneity_share()`: the relative size of the first-stage correction of PG26 eq. (29), averaged over coefficients. Values near zero mean the regressors behave as exogenous; large values mean the naive variance (eq. 27) would understate the uncertainty. It is computed from the robust variance whatever `vcov` the fit uses, and is `NA` for the benchmark estimators (see `ols_robust()`) and for `serial = "ma1"`, which do not compute it.

Usage

```
weight_diagnostics(fit)

ar_roots(fit)

nonexogeneity_share(fit)
```

Arguments

`fit`	An `"fgls"` object (with a heteroskedasticity correction for `weight_diagnostics()`, a serial-correlation correction otherwise).

Value

- `weight_diagnostics()`: a named numeric vector with elements `variance_ratio`, `effective_n_share` and `top1pct_weight_share`.
- `ar_roots()`: a complex vector with the k^* roots (empty when $k^* = 0$).
- `nonexogeneity_share()`: a single number.

See Also

Other diagnostics: `adf_test()`, `test_predetermined()`

Examples

```
fit <- fgls(y ~ x, data = simulate_combined(200, seed = 1), seed = 1)
weight_diagnostics(fit)
Mod(ar_roots(fit))
nonexogeneity_share(fit)
```

Index

∗ **benchmarks**
  ols_robust, 19
∗ **diagnostics**
  adf_test, 2
  test_predetermined, 29
  weight_diagnostics, 31
∗ **fgls**
  compare_fgls, 3
  fgls, 4
  fgls-methods, 11
  fgls_control, 14
  lasso_fgls, 16
  plot.fgls, 22
  serial_fgls, 23
  summary.fgls, 28
∗ **heteroskedasticity**
  lasso_fgls, 16
  make_dictionary, 18
∗ **simulation**
  simulate_serial, 25

adf_test, 2, *31, 32*
ar_roots *(*weight_diagnostics*)*, 31
as.Date(), *7*, *21*, *30*
augment.fgls *(*fgls-broom*)*, 9

cochrane_orcutt *(*ols_robust*)*, 19
coef(), *7*
compare_fgls, 3, *8*, *13*, *15*, *17*, *23*, *25*, *28*
compare_fgls(), *7*, *19*
confint(), *7*
confint.fgls *(*fgls-methods*)*, 11

df.residual.fgls *(*fgls-methods*)*, 11
durbin_estimator *(*ols_robust*)*, 19

fgls, *4*, 4, *13*, *15*, *17*, *23*, *25*, *28*
fgls(), *11*, *13, 14*, *16–18*, *22–24*
fgls-broom, 9
fgls-conditions, 10
fgls-methods, 11
fgls_control, *4*, *8*, *13*, 14, *17*, *23*, *25*, *28*
fgls_control(), *7*
fitted(), *5*, *16*, *21*, *24*, *30*
fitted.fgls *(*fgls-methods*)*, 11
formula.fgls *(*fgls-methods*)*, 11

glance.fgls *(*fgls-broom*)*, 9
glmnet::glmnet(), *15*

hybrid_gls *(*ols_robust*)*, 19

infeasible_gls *(*ols_robust*)*, 19

lasso_fgls, *4*, *8*, *13*, *15*, 16, *18*, *23*, *25*, *28*
lasso_fgls(), *4*, *8*, *11*, *13*, *18*

make_dictionary, *17*, 18
make_dictionary(), *6*, *15*, *17*
model.frame.fgls *(*fgls-methods*)*, 11
model.matrix.fgls *(*fgls-methods*)*, 11

nobs.fgls *(*fgls-methods*)*, 11
nonexogeneity_share
        *(*weight_diagnostics*)*, 31

ols_robust, 19
ols_robust(), *3*, *13*, *31*

plot(), *7*, *23*
plot.fgls, *4*, *8*, *13*, *15*, *17*, 22, *25*, *28*
predict(), *7*
predict.fgls *(*fgls-methods*)*, 11
print.fgls *(*fgls-methods*)*, 11
print.fgls_adf *(*adf_test*)*, 2
print.fgls_predetermined
        *(*test_predetermined*)*, 29
print.summary.fgls *(*summary.fgls*)*, 28

residuals(), *5*, *7*, *16*, *21*, *24*, *30*
residuals.fgls *(*fgls-methods*)*, 11

serial_fgls, *4*, *8*, *13*, *15*, *17*, *23*, 23, *28*
serial_fgls(), *4*, *8*, *11*, *13*, *21*
simulate_combined *(*simulate_serial*)*, 25
simulate_hetero *(*simulate_serial*)*, 25
simulate_predictive *(*simulate_serial*)*, 25
simulate_serial, 25
stats::lm(), *3*, *5*, *7*, *16*, *20, 21*, *24*, *29, 30*
stats::na.exclude(), *5*, *13*, *16*, *21*, *24*, *30*
stats::na.fail(), *5*, *16*, *20*, *24*, *30*
stats::na.omit(), *5*, *16*, *20*, *24*, *30*
stats::qqnorm(), *23*
summary(), *7*
summary.fgls, *4*, *8*, *13*, *15*, *17*, *23*, *25*, 28
summary.fgls(), *10*, *13*
suppressWarnings(), *10*

terms.fgls *(*fgls-methods*)*, 11
test_predetermined, *3*, 29, *32*
tidy.fgls *(*fgls-broom*)*, 9
tryCatch(), *10*

vcov(), *7*
vcov.fgls *(*fgls-methods*)*, 11
vcov.fgls(), *6*

weight_diagnostics, *3*, *31*, 31
withCallingHandlers(), *10*