Package 'ardldml'

September 15, 2026

Type Package

Title Bounds Testing for Cointegration with Many Persistent Controls

Version 0.1.0

Description An implementation of the DML-Bounds procedure of Villena (2026) <doi:10.2139/ssrn.6472826> for testing cointegration in data-rich time-series settings. The Autoregressive Distributed Lag (ARDL) bounds test of Pesaran, Shin and Smith (2001) <doi:10.1002/jae.616> avoids pretesting the integration order of the regressors but is not designed for a high-dimensional conditioning set. Residualising the lagged levels against persistent controls can absorb stochastic trends and thereby change the finite-sample null distribution, so what governs the null is the effective number of stochastic trends surviving residualisation rather than the integration order of the original regressors. The procedure combines h-block cross-fitting, a balanced nuisance projection in the Double Machine Learning (DML) style of Chernozhukov and others (2018) <doi:10.1111/ectj.12097>, adaptive weighting after Zou (2006) <doi:10.1198/016214506000000735>, and a restricted system wild bootstrap that regenerates the dependent variable and the focal regressor jointly. No critical-value table is shipped: the classical bracket is regenerated by simulation and the operational critical value is bootstrapped. A trend-absorption diagnostic and a penalty-sensitivity sweep report whether a verdict survives a change of conditioning set. Monthly United States macroeconomic series from the 'FRED-MD' database of McCracken and Ng (2016) <doi:10.1080/07350015.2015.1086655> are bundled so every example runs offline.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

Depends R (>= 4.1.0)

Imports glmnet, stats, graphics, grDevices, utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0), tseries

VignetteBuilder knitr

LazyData true

RoxygenNote 7.3.3

URL https://github.com/merwanroudane/ardldml

BugReports https://github.com/merwanroudane/ardldml/issues

Config/testthat/edition 3

NeedsCompilation no

Author Merwan Roudane [aut, cre, cph]

Maintainer Merwan Roudane <merwanroudane920@gmail.com>

Repository CRAN

Date/Publication 2026-09-15 12:20:09 UTC

Contents

`adaptive_post_lasso` *Adaptive Post-LASSO*

Description

LASSO selection followed by an unpenalised refit on the selected support, with optional adaptive weights on a designated block of columns.

Usage

```
adaptive_post_lasso(
  X,
  y,
  lam = NULL,
  adaptive = TRUE,
  c = 1.1,
  adaptive_mask = NULL,
  min_dof = 1L
)
```

Arguments

`X`	Numeric matrix of regressors.
`y`	Numeric target.
`lam`	Penalty. If `NULL`, `plugin_penalty` is used with a noise scale refined once from an initial fit.
`adaptive`	Enable adaptive weighting. `FALSE` gives plain LASSO.
`c`	Constant in the plug-in penalty.
`adaptive_mask`	Logical vector marking the columns that receive adaptive weights. If `NULL` and `adaptive` is `TRUE`, every column is weighted.
`min_dof`	Minimum residual degrees of freedom required for the refit.

Details

The adaptive weighting exists because vanilla ℓ_1 over-selects integrated regressors and thereby induces spurious trend absorption. Following the paper, the weights apply to the **integrated block only**; stationary columns stay under the plain penalty. The weight on column j is the absolute univariate slope of y on that column, applied by rescaling the column, which is equivalent to penalising $|\beta_j|$ by its reciprocal.

Selection runs on standardised columns so the penalty is scale-free, and the coefficients are mapped back before the refit. Degrees of freedom charge the selected support size, so a cell with non-positive residual degrees of freedom is reported as not estimable rather than silently returning a number.

Value

A list with `fitted`, `coef`, `support`, `intercept`, `lam` and `estimable`.

References

Zou, H. (2006). The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101(476), 1418–1429. doi:10.1198/016214506000000735

Examples

```
set.seed(1)
X <- matrix(rnorm(60 * 5), 60, 5)
y <- X[, 1] + rnorm(60)
fit <- adaptive_post_lasso(X, y, adaptive = FALSE)
which(fit$support)
```

`ardl_colors` *Colours Used by the Package's Figures*

Description

A colourblind-safe palette (Wong 2011) with semantic names, so the same contrast carries the same colour across every figure: the bootstrap against the borrowed bound, the full against the reduced control set.

Usage

```
ardl_colors
```

Format

A named character vector of hex colours.

Examples

```
ardl_colors[["bootstrap"]]
```

bounds_verdict *Mark a Statistic Against a Pair of Bounds*

Description

Returns "reject" above the upper bound, "inconclusive" between the two, and "fail to reject" below the lower bound. A single verdict would hide the inconclusive region, which is the whole point of a bounds test.

Usage

```
bounds_verdict(stat, lower, upper)
```

Arguments

stat	The statistic.
lower, upper	The bracket.

Value

A character string.

Examples

```
bounds_verdict(6.2, lower = 4.94, upper = 5.73)
bounds_verdict(5.2, lower = 4.94, upper = 5.73)
bounds_verdict(2.0, lower = 4.94, upper = 5.73)
```

build_balanced_design *Build the Balanced First-Stage Design*

Description

The two nuisance projections have different regressor sets, because their targets have different integration orders. ΔY_t is stationary, so the integrated controls enter its projection **in first differences**; regressing a stationary target on integrated levels would be unbalanced and spurious. The tested levels $Z_{t-1} = (Y_{t-1}, D_{t-1})$ are integrated and are projected on the control **levels**. That second projection is the only place trend absorption can happen, and therefore the only place the effective integrated count is determined.

Usage

```
build_balanced_design(
  y,
  d,
  W,
  lags = 4L,
  integrated = NULL,
  dlags = FALSE,
  adf_alpha = 0.1
)

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

Arguments

y	Numeric vector: the outcome in levels.
d	Numeric vector: the focal regressor in levels.
W	Matrix or data frame of controls in levels, with column names.
lags	Short-run lag order p: how many lagged differences of y (and of d, if dlags) enter the stationary design.
integrated	Character vector of controls to treat as integrated of order one; see classify_controls.
dlags	Include lags of ΔD in the conditional design. Defaults to FALSE, which is the paper's equation (3) as written: it carries the contemporaneous ΔD_t term and lagged ΔY only. TRUE gives the general ARDL(p, q) short-run structure, which is standard practice but is not what the paper specifies. The bootstrap's marginal model keeps its own lags of ΔD either way.
adf_alpha	Level for the Augmented Dickey-Fuller fallback.
x	An "ardl_design" object.
...	Ignored.

Details

A stationary regressor cannot track the stochastic trend of an integrated one, so partialling out stationary controls leaves the unit root intact no matter how many there are. Only integrated controls can absorb a trend.

Value

An object of class "ardl_design": a list with dY, X and its column groups, Z, Wlev, dD_lags, index, stationary and integrated.

Examples

```
df <- passthrough_regime("1999-2007")
des <- build_balanced_design(df$cpi, df$neer,
                             as.matrix(df[, CONTROLS]),
```

```
                              lags = 4, integrated = DEFAULT_INTEGRATED)
des
```

CASE_LABELS *Deterministic Case Labels*

Description

The five deterministic specifications of Pesaran, Shin and Smith (2001). Cases 2 and 4 restrict the intercept or the trend, which puts it inside the tested null and adds one restriction; see `n_restrictions`.

Usage

```
CASE_LABELS
```

Format

A named character vector of length 5.

References

Pesaran, M. H., Shin, Y. and Smith, R. J. (2001). Bounds testing approaches to the analysis of level relationships. *Journal of Applied Econometrics*, 16(3), 289–326. doi:10.1002/jae.616

`classical_bounds_test` *The Classical Bounds Test*

Description

All three steps of the Pesaran, Shin and Smith procedure. Rejecting the joint F null is not on its own evidence of a level relationship, because two degenerate cases survive it, so the t test on the speed of adjustment and the Wald test on the long-run coefficients are reported alongside.

Usage

```
classical_bounds_test(
  y,
  x,
  lags = 1L,
  order = 1L,
  case = 3L,
  fixed = NULL,
  nsim = 20000L,
  seed = 0,
  levels = DEFAULT_LEVELS
)
```

Arguments

`y`	Numeric vector: outcome in levels.
`x`	Matrix or data frame of long-run forcing regressors in levels.
`lags`	ARDL order p.
`order`	ARDL order q.
`case`	Deterministic case, an integer in 1:5. Cases 2 and 3 estimate the same model and differ only in whether the intercept is inside the tested null; the same holds for the trend in cases 4 and 5.
`fixed`	Optional matrix of regressors entered contemporaneously and never lagged – the z_t of the ARDL literature. They shift short-run dynamics but are excluded from the long-run relationship, which is what you want for a variable that predicts the outcome but cannot plausibly cointegrate with it.
`nsim`	Replications for the simulated bracket.
`seed`	Seed for the bracket simulation.
`levels`	Significance levels for the bracket.

Details

The bracket is generated by `simulate_pss_bounds` at the model's own sample size, not read off a table calibrated at $T = 1000$.

Value

An object of class `"classical_bounds"` with the F and t statistics, the step-3 Wald statistic and its p-value, the speed of adjustment, the long-run coefficients and the simulated bracket.

References

Pesaran, M. H., Shin, Y. and Smith, R. J. (2001). Bounds testing approaches to the analysis of level relationships. *Journal of Applied Econometrics*, 16(3), 289–326. doi:10.1002/jae.616

Examples

```
df <- passthrough_regime("1999-2007")
# Small nsim keeps this quick; raise it for anything you report.
cb <- classical_bounds_test(df$cpi, cbind(neer = df$neer), lags = 4,
                            nsim = 100, seed = 1)
cb
```

classify_controls *Split the Control Set into Stationary and Integrated Blocks*

Description

The balanced design has to know which controls to difference, so the two blocks must be named. Passing `integrated` explicitly, on economic grounds, is the recommended route: the appeal of the bounds framework is that it avoids pretesting, and a pretest here would put its own error into everything downstream without any of the inference accounting for it.

Usage

```
classify_controls(W, integrated = NULL, alpha = 0.1)
```

Arguments

`W`	A matrix or data frame of controls in levels, with column names.
`integrated`	Character vector of control names to treat as integrated of order one. If supplied, the classification is taken as stated.
`alpha`	Significance level for the Augmented Dickey-Fuller fallback.

Details

If `integrated` is `NULL` an Augmented Dickey-Fuller fallback is used, which requires the **tseries** package.

Value

A list with character vectors `stationary` and `integrated`.

Examples

```
W <- as.matrix(passthrough[, c("m2", "unrate")])
classify_controls(W, integrated = "m2")
```

compute_statistic *Build the Design, Cross-Fit, Residualise and Test*

Description

The whole pipeline in one call, so the bootstrap can invoke it on a regenerated path and obtain a statistic carrying the same generated-regressor error as the observed one.

Usage

```
compute_statistic(y, d, W, spec, frozen_dY_support = NULL)
```

Arguments

y	Numeric vector: outcome in levels.
d	Numeric vector: focal regressor in levels.
W	Matrix or data frame of controls in levels.
spec	A dml_spec.
frozen_dY_support	Optional logical vector fixing the support of the stationary projection. The paper freezes the stationary support across bootstrap draws while re-selecting the level supports, so that selection error is reflected in the bootstrap law.

Value

A list with stat, alpha, theta, theta_se, design, folds, supports, resid and estimable.

Examples

```
df <- passthrough_regime("1999-2007")
out <- compute_statistic(df$cpi, df$neer, as.matrix(df[, CONTROLS]),
                         dml_spec(lags = 4, n_blocks = 5, buffer = 6,
                                  integrated = DEFAULT_INTEGRATED))
round(out$stat, 4)
```

---

conditional_ecm *Fit the Unrestricted Conditional Error Correction Model*

Description

The classical half of the package: the unrestricted conditional error correction model of Pesaran, Shin and Smith (2001),

$$\Delta y_t = c_0 + c_1 t + \pi_y y_{t-1} + \pi_x' x_{t-1} + \sum_{i=1}^{p-1} \psi_{yi} \Delta y_{t-i} + \sum_{i=0}^{q-1} \psi_{xi}' \Delta x_{t-i} + \gamma' z_t + u_t,$$

from which the speed of adjustment is $\alpha = -\pi_y$ and the long-run coefficients are $\theta = \pi_x / \alpha$.

Usage

```
conditional_ecm(y, x, lags = 1L, order = 1L, case = 3L, fixed = NULL)
```

Arguments

y	Numeric vector: outcome in levels.
x	Matrix or data frame of long-run forcing regressors in levels.
lags	ARDL order p.
order	ARDL order q.
case	Deterministic case, an integer in 1:5. Cases 2 and 3 estimate the same model and differ only in whether the intercept is inside the tested null; the same holds for the trend in cases 4 and 5.
fixed	Optional matrix of regressors entered contemporaneously and never lagged – the z_t of the ARDL literature. They shift short-run dynamics but are excluded from the long-run relationship, which is what you want for a variable that predicts the outcome but cannot plausibly cointegrate with it.

Value

A list with the design matrix X, the response dy, coefficients, vcov, nobs and the column names of the level terms.

References

Pesaran, M. H., Shin, Y. and Smith, R. J. (2001). Bounds testing approaches to the analysis of level relationships. *Journal of Applied Econometrics*, 16(3), 289–326. doi:10.1002/jae.616

Examples

```
df <- passthrough_regime("1999-2007")
m <- conditional_ecm(df$cpi, cbind(neer = df$neer), lags = 4, case = 3)
m$nobs
```

CONTROLS *The Seven Controls of the Full Conditioning Set*

Description

The Seven Controls of the Full Conditioning Set

Usage

```
CONTROLS
```

Format

A character vector of length 7.

`cross_fit_projection` *Cross-Fitted Projection*

Description

Out-of-fold predictions of `y` on `X` under an h-block partition. Each evaluation block is predicted by a model estimated on that block's buffered training set, so the first-stage error is decoupled from the evaluation-fold innovations.

Usage

```
cross_fit_projection(
  X,
  y,
  folds,
  lam = NULL,
  adaptive = TRUE,
  c = 1.1,
  penalised = TRUE,
  adaptive_mask = NULL
)
```

Arguments

`X`	Numeric matrix of regressors.
`y`	Numeric target.
`folds`	An `"ardl_folds"` object from `hblock_folds`.
`lam`	Penalty passed to `adaptive_post_lasso`.
`adaptive`	Adaptive weighting.
`c`	Constant in the plug-in penalty.
`penalised`	If `FALSE`, the projection is unpenalised least squares: the low-dimensional corner, and the `"ols"` arm of the trend-absorption diagnostic.
`adaptive_mask`	Columns receiving adaptive weights.

Value

A list with `fitted`, `resid`, `support_union`, `lams` and `estimable`.

Examples

```
set.seed(1)
X <- matrix(rnorm(80 * 3), 80, 3)
y <- X[, 1] + rnorm(80)
fit <- cross_fit_projection(X, y, hblock_folds(80, 4, 2), penalised = FALSE)
stats::sd(fit$resid)
```

decision *Verdict of a Fitted Test*

Description

Verdict of a Fitted Test

Usage

```
decision(object, level = 0.05)
```

Arguments

object	A "dml_bounds" object.
level	Significance level.

Value

"reject", "fail to reject", or "no bootstrap run".

Examples

```
df <- passthrough_regime("1999-2007")
fit <- dml_bounds(df$cpi, df$neer, as.matrix(df[, CONTROLS]),
                  lags = 4, n_blocks = 5, buffer = 6,
                  integrated = DEFAULT_INTEGRATED)
decision(fit)
```

DEFAULT_INTEGRATED *Controls Treated as Integrated of Order One*

Description

On economic grounds, not by pretest.

Usage

```
DEFAULT_INTEGRATED
```

Format

A character vector of length 6.

DEFAULT_LEVELS *Significance Levels Reported by Default*

Description

Significance Levels Reported by Default

Usage

```
DEFAULT_LEVELS
```

Format

A numeric vector of length 3.

dml_bootstrap *The Restricted System Wild Bootstrap*

Description

Attaches a critical value and a p-value to a fitted `dml_bounds` object. This is the inference: because the tabulated bounds are not operationally valid once the conditioning set is large and persistent, the critical value is generated from the data under the imposed null.

Usage

```
dml_bootstrap(
  object,
  B = 999L,
  level = 0.05,
  seed = NULL,
  scheme = c("system", "fixed"),
  freeze_stationary_support = TRUE,
  progress = FALSE
)
```

Arguments

`object`	A fitted `"dml_bounds"` object.
`B`	Bootstrap replications. The paper uses 999.
`level`	Significance level for the reported critical value.
`seed`	Optional seed.
`scheme`	`"system"` for Algorithm 1, or `"fixed"` for the strong-exogeneity special case.
`freeze_stationary_support`	Freeze the stationary first-stage support across draws while re-selecting the level supports, as the paper does.
`progress`	Print a line every 10% of draws.

Details

Two auxiliary models are estimated under H_0: a restricted conditional model for ΔY_t carrying the same deterministic terms and short-run lag structure as the empirical specification but excluding the lagged levels and the confounder levels; and a marginal model for ΔD_t on an intercept, its own lags and the first-stage-selected differenced controls. A *single* Rademacher sequence is then applied to the **stacked** residual pair, both paths are regenerated recursively, and the entire residualised statistic is recomputed with the level supports re-selected.

Sharing one weight is what keeps the endogeneity channel alive. The Pesaran-Shin-Smith framework exists *because* the focal regressor need not be exogenous, and the conditional model absorbs the contemporaneous correlation through the ΔD_t term. A scheme that holds `d` at its realised path and reweights the equation error alone simulates a world with zero correlation between the two innovations whatever the data say; it is available as `scheme = "fixed"` and is valid only under strong exogeneity.

The controls are held at their realised path, which conditions the bootstrap on the realised trend content. Because the marginal model conditions on the differenced controls, any stochastic trend that `d` shares with `W` is inherited rather than broken.

Value

The object with a `boot` component attached, holding `crit`, `pvalue`, `draws`, `B`, `level`, `scheme`, `n_failed` and `corr_eps_v`.

References

Villena, M. J. (2026). Testing cointegration with many persistent controls. SSRN working paper. doi:10.2139/ssrn.6472826

Examples

```
df <- passthrough_regime("1999-2007")
fit <- dml_bounds(df$cpi, df$neer, as.matrix(df[, CONTROLS]),
                  lags = 4, n_blocks = 5, buffer = 6,
                  integrated = DEFAULT_INTEGRATED)
# Tiny B so the example is quick; use B = 999 for anything you report.
fit <- dml_bootstrap(fit, B = 5, seed = 1)
fit$boot$pvalue
```

`dml_bounds` *Test for a Long-Run Relationship with Many Persistent Controls*

Description

Fits the DML-Bounds test of Villena (2026): the lagged levels $Z_{t-1} = (Y_{t-1}, D_{t-1})$ are orthogonalised against a high-dimensional control set with a balanced, cross-fitted first stage, and the null of no level relationship is tested on the residualised variables.

Usage

```
dml_bounds(y, d, W, B = 0L, level = 0.05, seed = NULL, ...)

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

## S3 method for class 'dml_bounds'
summary(object, ...)
```

Arguments

y	Numeric vector: outcome in levels.
d	Numeric vector: focal regressor in levels.
W	Matrix or data frame of controls in levels, with column names.
B	Bootstrap replications. 0 fits only, which is fast and is what the short examples use.
level	Significance level for the reported critical value.
seed	Optional seed for the bootstrap.
...	Passed to dml_spec.
x, object	A "dml_bounds" object.

Details

Do **not** compare the statistic with a tabulated bound. The asymptotic reference sits somewhere inside the Pesaran-Shin-Smith bracket, at a position governed by the number of stochastic trends that survive residualisation, and the finite-sample law is further perturbed by generated-regressor error. Call dml_bootstrap, or pass B, to attach a critical value.

The estimand is conditional. This does not ask whether y and d cointegrate unconditionally; it asks whether they cointegrate *given* W. If a control is itself part of the equilibrium system, partialling it out removes the relation rather than the confounding, and a non-rejection then reflects over-absorption. That failure mode is not visible in a single fit – use trend_absorption.

Value

An object of class "dml_bounds".

References

Villena, M. J. (2026). Testing cointegration with many persistent controls. SSRN working paper. doi:10.2139/ssrn.6472826

See Also

dml_bootstrap, trend_absorption, classical_bounds_test

Examples

```
df <- passthrough_regime("1999-2007")
fit <- dml_bounds(df$cpi, df$neer, as.matrix(df[, CONTROLS]),
                  lags = 4, n_blocks = 5, buffer = 6,
                  integrated = DEFAULT_INTEGRATED)
fit


# With inference. The paper uses B = 999; this is smaller so it stays quick.
fit <- dml_bootstrap(fit, B = 99, seed = 20260625)
summary(fit)
```

---

dml_spec *Specification of a DML-Bounds Fit*

Description

Everything that defines a fit, held separately from the data so the bootstrap can rebuild the identical statistic on a regenerated path.

Usage

```
dml_spec(
  lags = 4L,
  case = 3L,
  n_blocks = 5L,
  buffer = 0L,
  adaptive = TRUE,
  penalised = TRUE,
  penalty = "plugin",
  c = 1.1,
  integrated = NULL,
  include_constant = FALSE,
  adaptive_integrated_only = TRUE,
  dlags = FALSE
)

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

Arguments

lags	Short-run lag order p.
case	Deterministic case, 1 or 3. This does **not** enter the statistic, which is a no-intercept projection on the orthogonalised levels regardless. It sets the deterministic terms of the restricted conditional model the bootstrap resamples from.

n_blocks	Cross-fitting blocks K.
buffer	Cross-fitting buffer h.
adaptive	Adaptive weights on the level projection. TRUE is the paper's default; FALSE gives the plain-LASSO arm of the diagnostic.
penalised	If FALSE, both projections are unpenalised least squares – the low-dimensional corner and the "ols" arm.
penalty	How the ΔY penalty is chosen: "plugin" (the default), one of "low", "medium", "high" (equivalently "min", "mid", "1se") for rolling-origin cross-validation, or a number fixing it directly. The level projection always uses the plug-in rule.
c	Constant in the plug-in penalty.
integrated	Character vector of controls to treat as integrated of order one.
include_constant	Add an intercept to the final regression. Defaults to FALSE, which is the paper's equation (10) as written; the first-stage projections already carry intercepts, so the residuals are mean-zero by construction. Exposed only for sensitivity checks.
adaptive_integrated_only	Restrict the adaptive weights to the integrated block, which is what the paper specifies.
dlags	Include lags of ΔD in the conditional design.
x	A "dml_spec" object.
...	Ignored.

Value

An object of class "dml_spec".

Examples

```
dml_spec(lags = 4, n_blocks = 5, buffer = 6)
```

---

hblock_folds *h-Block Cross-Fitting Folds*

Description

Cuts the sample into n_blocks contiguous chronological blocks and builds, for each, a training set that excludes the block itself and the buffer observations either side of it.

Usage

```
hblock_folds(n, n_blocks = 5L, buffer = 0L)

## S3 method for class 'ardl_folds'
as.data.frame(x, ...)

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

Arguments

`n`	Number of observations.
`n_blocks`	Number of chronological blocks K; at least 2.
`buffer`	Buffer width h in observations.
`x`	An `"ardl_folds"` object.
`...`	Ignored.

Details

In cross-sectional double machine learning the folds are random. In a time series they cannot be: randomly held-out observations are adjacent in time to their own training data, so the first-stage error stays correlated with the evaluation innovations. The buffer is what buys the decoupling, and it costs sample – at most $h(K-1)$ observations across the training sets. Use `sample_use_table` to see the cost before choosing a configuration.

Value

An object of class `"ardl_folds"`: a list with `n`, `n_blocks`, `buffer`, and lists `eval` and `train` of integer index vectors.

See Also

`sample_use`, `sample_use_table`

Examples

```
f <- hblock_folds(100, n_blocks = 4, buffer = 5)
sapply(f$train, length)
```

k_convention_offset *The k-Convention Offset*

Description

Two conventions for k sit next to each other in this literature, and mixing them shifts every bound by one row. Pesaran, Shin and Smith count the long-run forcing regressors only; some implementations pass the number of level terms, which is one larger, into a table indexed the first way. Every bound then comes out one row too far down, hence too small, and the test over-rejects.

Usage

```
k_convention_offset(k = 1L, case = 3L, level = 0.05)
```

Arguments

k	Number of long-run forcing regressors; see n_restrictions.
case	Deterministic case, an integer in 1:5.
level	Significance level to report.

Details

This function puts the correct row next to the shifted one so the size of the distortion can be seen. It is a demonstration, not something the package uses internally: classical_bounds_test indexes on k.

Value

A data frame with one row for the correct bounds and one for the shifted bounds.

Examples

```
k_convention_offset(k = 1, case = 3)
```

LOG_SERIES *Series Transformed to Logs*

Description

The quantity and price series. The four interest and unemployment rates stay in levels: they are already in percentage-point units and can approach zero.

Usage

```
LOG_SERIES
```

Format

A character vector of length 5.

`n_restrictions` *Number of Restrictions in the Bounds Test*

Description

The level terms are y_{t-1} and the k lagged forcing regressors. Cases 2 and 4 add one more because the intercept or the trend is restricted and therefore tested jointly with them.

Usage

```
n_restrictions(k, case = 3L)
```

Arguments

`k`	Number of long-run forcing regressors. This does *not* count the dependent variable; mixing that convention up shifts every bound by one row.
`case`	Deterministic case, an integer in 1:5.

Value

An integer.

Examples

```
n_restrictions(k = 1, case = 3)
n_restrictions(k = 1, case = 4)
```

`parula_colors` *MATLAB Parula Color Palette*

Description

The MATLAB R2014b Parula colormap as hexadecimal colors, interpolated from its 64 stops. Parula is perceptually uniform and runs from dark blue-purple through teal and green to yellow.

Usage

```
parula_colors(n = 256)
```

Arguments

`n`	Number of colors to interpolate.

Value

A character vector of `n` hex colors.

Examples

```
parula_colors(6)
```

`passthrough` *Exchange-Rate Pass-Through Data*

Description

Nine monthly United States macroeconomic series, 1973-01 to 2020-12, in raw levels. These are the series behind the paper's main application, bundled so every example runs offline.

Usage

```
passthrough
```

Format

A data frame with 576 rows and 10 columns:

date First day of the month, class `Date`.

cpi Consumer price index, all urban consumers (`CPIAUCSL`).

neer Trade-weighted United States dollar index (`TWEXAFEGSMTHx`).

m2 M2 money stock (`M2SL`).

ffr Effective federal funds rate (`FEDFUNDS`).

ip Industrial production index (`INDPRO`).

unrate Civilian unemployment rate (`UNRATE`).

oil Crude oil spot price, West Texas Intermediate (`OILPRICEx`).

gs10 Ten-year Treasury constant maturity yield (`GS10`).

baa Moody's Baa corporate bond yield (`BAA`).

Details

The sample starts in 1973-01 because that is the first observation of the trade-weighted dollar index, which is also why the first monetary regime starts there. The four regimes give complete samples of 156, 156, 108 and 156 observations.

Source

FRED-MD, the monthly macroeconomic database of McCracken and Ng (2016), vintage 2025-06. doi:10.1080/07350015.2015.1086655

References

McCracken, M. W. and Ng, S. (2016). FRED-MD: A monthly database for macroeconomic research. *Journal of Business & Economic Statistics*, 34(4), 574–589. doi:10.1080/07350015.2015.1086655

Examples

```
str(passthrough)
range(passthrough$date)
```

`passthrough_regime` *Load a Regime of the Pass-Through Data*

Description

Load a Regime of the Pass-Through Data

Usage

```
passthrough_regime(regime = NULL, log = TRUE, start = NULL, end = NULL)
```

Arguments

`regime`	One of the names of `PASSTHROUGH_REGIMES`, or `NULL` for the full 1973-2020 sample.
`log`	Take logs of `LOG_SERIES`.
`start`, `end`	Optional `"YYYY-MM"` bounds, applied after `regime`.

Value

A data frame with a `date` column and the nine series.

Examples

```
df <- passthrough_regime("1999-2007")
nrow(df)
head(df[, c("date", "cpi", "neer")])
```

PASSTHROUGH_REGIMES *The Four Monetary Regimes*

Description

Testing within regimes rather than across them avoids conflating a structural break with a long-run relationship.

Usage

```
PASSTHROUGH_REGIMES
```

Format

A named list of start and end months, as `"YYYY-MM"` strings.

PENALTY_RULES *Penalty Rules of the Robustness Grid*

Description

The three penalty choices the paper's robustness tables label Low, Medium and High, mapped onto the points of the cross-validation profile they take.

Usage

```
PENALTY_RULES
```

Format

A named character vector of length 3.

`penalty_sensitivity` *Sweep the Penalty and the Projection*

Description

The paper's robustness tables vary the penalty, the lag order and the level projection, and a verdict can turn on that choice. Reporting a single cell from this grid is specification search; reporting the grid is the method.

Usage

```
penalty_sensitivity(
  y,
  d,
  W,
  rules = c("low", "medium", "high"),
  lags_grid = 4L,
  projections = c("adaptive", "plain", "ols"),
  B = 0L,
  seed = NULL,
  ...
)
```

Arguments

`y`	Numeric vector: outcome in levels.
`d`	Numeric vector: focal regressor in levels.
`W`	Matrix or data frame of controls in levels.
`rules`	Penalty rules to sweep: `"low"`, `"medium"`, `"high"`.
`lags_grid`	Short-run lag orders to try.
`projections`	Estimators to compare: `"adaptive"` (the paper's), `"plain"` (vanilla ℓ_1) and `"ols"` (the unpenalised benchmark).
`B`	Bootstrap draws per cell, or `0` for statistics only, which is much faster.
`seed`	Base seed; each fit is offset so the four bootstraps are independent.
`...`	Passed to `dml_spec`; `adaptive` is set per fit.

Details

The column to watch is `n_selected_Z`, the number of control **levels** the level projection retained. It is the empirical counterpart of the effective integrated count: zero means nothing was absorbed, so the test sits at the classical corner and orthogonalisation did nothing, while a large value means heavy absorption and the long-run coefficient should be checked for instability. A long-run coefficient that changes sign across the grid is a warning that the conditioning set, not the data, is driving the answer.

Value

A data frame with one row per cell, carrying a "theta_sign_flips" attribute.

Examples

```
df <- passthrough_regime("1999-2007")
penalty_sensitivity(df$cpi, df$neer, as.matrix(df[, CONTROLS]),
                    lags_grid = 4, n_blocks = 5, buffer = 6,
                    integrated = DEFAULT_INTEGRATED)
```

---

plot.dml_bounds *Plot the Simulated Null of a Fitted Test*

Description

The bootstrap null with the observed statistic and, optionally, a borrowed classical bound marked. The gap between the bootstrap critical value and the borrowed bound is the whole argument for not reading the statistic against a table.

Usage

```
## S3 method for class 'dml_bounds'
plot(x, borrowed = 5.73, breaks = 30, main = "Bootstrap null", xlab = "F", ...)
```

Arguments

x	A "dml_bounds" object that has been bootstrapped.
borrowed	Optional classical bound to mark, for contrast.
breaks	Passed to hist.
main, xlab	Plot labels.
...	Passed to hist.

Value

Invisibly, the object.

Examples

```
df <- passthrough_regime("1999-2007")
fit <- dml_bounds(df$cpi, df$neer, as.matrix(df[, CONTROLS]),
                  lags = 4, n_blocks = 5, buffer = 6,
                  integrated = DEFAULT_INTEGRATED, B = 5, seed = 1)
plot(fit)
```

plot_bracket *Plot the Trend-Absorption Bracket*

Description

Where the limiting null sits as a function of the number of stochastic trends that survive residualisation. As that count falls from k to zero, the null slides from the integrated endpoint to the stationary one; classical bounds testing is the right-hand end of this picture.

Usage

```
plot_bracket(
  k = 10L,
  lower = 4.94,
  upper = 5.73,
  k_tilde = NULL,
  main = "Trend-absorption bracket",
  xlab = "effective integrated count",
  ylab = "limiting null, 95th percentile"
)
```

Arguments

k	Number of level terms.
lower, upper	The classical bracket endpoints.
k_tilde	Optional effective count to mark.
main, xlab, ylab	Plot labels.

Value

Invisibly, a data frame of the plotted curve.

Examples

```
plot_bracket(k = 10, k_tilde = 6)
```

plugin_penalty *The Plug-In Penalty*

Description

$\lambda = c\sqrt{\log(d)/n}\hat{\sigma}$, the standard high-dimensional choice: large enough to dominate the noise, small enough to leave the signal. The paper uses $c = 1.1$.

Usage

```
plugin_penalty(n, d, sigma, c = 1.1)
```

Arguments

n	Sample size.
d	Number of regressors.
sigma	Noise scale.
c	Constant.

Value

A single number.

Examples

```
plugin_penalty(n = 100, d = 40, sigma = 1)
```

---

print.classical_bounds

Print a Classical Bounds Test

Description

Print a Classical Bounds Test

Usage

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

Arguments

x	A "classical_bounds" object from classical_bounds_test.
...	Ignored.

Value

Invisibly, x.

Examples

```
df <- passthrough_regime("1999-2007")
print(classical_bounds_test(df$cpi, cbind(neer = df$neer), lags = 4,
                            nsim = 100, seed = 1))
```

pss_reference *Published Bounds for a Validated Cell*

Description

A small set of cells from the printed Pesaran, Shin and Smith (2001) tables, stored so `simulate_pss_bounds` can be checked against print and so the borrowed-bound comparison can quote the exact published number. Only a subset of cells is stored; the simulator has no such gaps.

Usage

```
pss_reference(k, case = 3L)
```

Arguments

`k`	Number of long-run forcing regressors; see `n_restrictions`.
`case`	Deterministic case, an integer in 1:5.

Value

A data frame with columns `level`, `I0` and `I1`.

Examples

```
pss_reference(k = 1, case = 3)
```

rademacher *Rademacher Weights*

Description

A vector of independent ± 1 draws.

Usage

```
rademacher(n)
```

Arguments

`n`	Length.

Value

A numeric vector of `-1` and `1`.

Examples

```
set.seed(1)
rademacher(8)
```

`REDUCED_DROP` *Controls Dropped to Form the Reduced Set*

Description

The two most likely to share a stochastic trend with the pass-through relation itself, which is what makes them the interesting ones to drop in the trend-absorption diagnostic.

Usage

```
REDUCED_DROP
```

Format

A character vector of length 2.

`sample_use` *Share of the Sample Available for Training*

Description

Average, across blocks, of the training-set size as a fraction of `n`. With no buffer this is $1 - 1/K$; each unit of h removes roughly $2h/n$ more. This is the quantity that makes the cost of buffering concrete.

Usage

```
sample_use(n, n_blocks = 5L, buffer = 0L)
```

Arguments

`n`	Number of observations.
`n_blocks`	Number of chronological blocks K; at least 2.
`buffer`	Buffer width h in observations.

Value

A single number between 0 and 1.

Examples

```
sample_use(108, n_blocks = 5, buffer = 6)
```

sample_use_table *Training Share Across a Grid of K and h*

Description

Tabulates `sample_use` over a grid, so the point at which the buffer costs more than the cross-fitting buys is visible before a configuration is committed to. Infeasible cells are `NA`.

Usage

```
sample_use_table(n, n_blocks = c(4L, 5L, 6L, 8L), buffers = c(0L, 2L, 5L, 10L))
```

Arguments

`n`	Number of observations.
`n_blocks`	Integer vector of block counts K.
`buffers`	Integer vector of buffer widths h.

Value

A numeric matrix, rows named by K and columns by h.

Examples

```
sample_use_table(108)
```

simulate_pss_bounds *Generate the Classical Bounds by Simulation*

Description

Regenerates the Pesaran, Shin and Smith (2001) bracket from the data-generating process printed in the notes to their Table CI, rather than reading a stored table.

Usage

```
simulate_pss_bounds(
  k,
  case = 3L,
  TT = 1000L,
  nsim = 40000L,
  levels = DEFAULT_LEVELS,
  seed = NULL
)
```

Arguments

k	Number of long-run forcing regressors; see n_restrictions.
case	Deterministic case, an integer in 1:5.
TT	Sample size. TT = 1000 reproduces the published asymptotic tables; smaller values give finite-sample bounds.
nsim	Replications. The published tables use 40000. Tail quantiles settle slowest, so reduce this only for exploratory work.
levels	Significance levels.
seed	Optional seed. Passed to set.seed; the caller's random number state is restored on exit.

Details

Under the null, with $y_0 = x_0 = 0$ and $k + 1$ independent standard normal innovations, $y_t = y_{t-1} + \epsilon_{1t}$ and $x_t = Px_{t-1} + e_{2t}$, with $P = I_k$ for the upper bound (regressors purely integrated of order one) and $P = 0$ for the lower bound (purely stationary). The statistic is the F form of the Wald test on the level terms.

Generating rather than tabulating has three advantages: it is checkable against print via pss_reference; it extends to any sample size, so small-sample bounds come from passing TT = n rather than shipping a second table; and it extends past $k = 10$, where the published tables stop.

Value

A data frame with columns level, I0 (the lower, stationary bound) and I1 (the upper, integrated bound).

References

Pesaran, M. H., Shin, Y. and Smith, R. J. (2001). Bounds testing approaches to the analysis of level relationships. *Journal of Applied Econometrics*, 16(3), 289–326. doi:10.1002/jae.616

Narayan, P. K. (2004). Reformulating critical values for the bounds F-statistics approach to cointegration. Monash University Discussion Paper 02/04.

Examples

```
# Small and fast; raise nsim for anything you intend to quote.
simulate_pss_bounds(k = 1, case = 3, TT = 120, nsim = 100, seed = 1)


# Closer to the published asymptotic table.
simulate_pss_bounds(k = 1, case = 3, TT = 1000, nsim = 4000, seed = 1)
```

`trend_absorption` *The Trend-Absorption Diagnostic*

Description

The most important thing in the package to actually run, and the easiest to skip.

Usage

```
trend_absorption(y, d, W, drop, B = 999L, seed = NULL, progress = FALSE, ...)

## S3 method for class 'trend_absorption'
as.data.frame(x, ...)

absorption_verdict(x, level = 0.05)

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

Arguments

`y`	Numeric vector: outcome in levels.
`d`	Numeric vector: focal regressor in levels.
`W`	Matrix or data frame of controls in levels.
`drop`	Character vector of controls to omit from the reduced set. Choose these on *economic* grounds: the ones most likely to be cointegrated with the tested relation itself.
`B`	Bootstrap replications per fit. Four bootstraps are run, so the cost is roughly `4 * B` statistic evaluations.
`seed`	Base seed; each fit is offset so the four bootstraps are independent.
`progress`	Report which fit is running.
`...`	Passed to `dml_spec`; `adaptive` is set per fit.
`x`	A `"trend_absorption"` object.
`level`	Significance level.

Details

Residualisation is valid only when the controls remove *nuisance* stochastic trends, not the equilibrium relation being tested. If a control is itself part of the equilibrium system, partialling it out absorbs the very trend the test is meant to detect, and a non-rejection then reflects over-absorption rather than the absence of a long-run relationship. That requirement cannot be verified directly; this contrast is the practical substitute.

Four fits are run: the full and reduced control sets crossed with the adaptive and the unpenalised level projection. Writing p_{ad} and p_{ols} for the bootstrap p-values under the two projections and p_{full}

and p_{red} for those under the two control sets, the diagnostic is the pair of gaps $\Delta_m = p_{ols} - p_{ad}$ and $\Delta_W = p_{full} - p_{red}$, read together with the stability of the long-run coefficient.

A large positive Δ_W – a verdict that rejects under the reduced set but not under the full set – together with a long-run coefficient that is more sharply estimated under the reduced fit, is evidence that the nuisance space is absorbing part of the tested relation, and the reduced-set verdict is then the more credible one. Concordant verdicts across the four fits indicate the conclusion is not an artefact of over-absorption.

This is a hypothesis-generating device, not a formal test. It has no size and no power; it tells you where to look.

Value

An object of class `"trend_absorption"`.

References

Villena, M. J. (2026). Testing cointegration with many persistent controls. SSRN working paper. doi:10.2139/ssrn.6472826

Examples

```
df <- passthrough_regime("1999-2007")
ta <- trend_absorption(df$cpi, df$neer, as.matrix(df[, CONTROLS]),
                       drop = REDUCED_DROP, B = 19, seed = 1,
                       lags = 4, n_blocks = 5, buffer = 6,
                       integrated = DEFAULT_INTEGRATED)
ta
```

---

tscv_penalty *Rolling-Origin Cross-Validation for the Penalty*

Description

For origins $t = T_0, \ldots, T-1$ the first stage is fitted on $\{1, \ldots, t\}$ and evaluated one step ahead, and λ minimises the average out-of-sample squared error. This respects temporal ordering, unlike ordinary k-fold cross-validation, which would train on the future.

Usage

```
tscv_penalty(
  X,
  y,
  grid = NULL,
  min_train = NULL,
  adaptive = FALSE,
  n_grid = 20L,
```

```
  rule = "min",
  return_profile = FALSE
)
```

Arguments

X	Numeric matrix of regressors.
y	Numeric target.
grid	Penalties to search. Defaults to a log grid spanning two decades around the plug-in value.
min_train	T_0. Defaults to max(20, n %/% 3).
adaptive	Adaptive weights during the search.
n_grid	Grid size when grid is NULL.
rule	One of "min" (Low), "1se" (High) or "mid" (Medium, the geometric midpoint); the labels "low", "medium" and "high" are accepted too.
return_profile	Also return the grid and its error profile.

Details

The verdict can depend on which point of the profile is taken, which is why penalty_sensitivity sweeps all three rather than reporting one.

Value

A single penalty, or a list with the profile if return_profile = TRUE.

Examples

```
set.seed(1)
X <- matrix(rnorm(60 * 4), 60, 4)
y <- X[, 1] + rnorm(60)
tscv_penalty(X, y, n_grid = 4, min_train = 45)
```

Index