Package 'ARDL'

May 10, 2026

Type Package

Title ARDL, ECM and Bounds-Test for Cointegration

Description Creates complex autoregressive distributed lag (ARDL) models and constructs the underlying unrestricted and restricted error correction model (ECM) automatically, just by providing the order. It also performs the bounds-test for cointegration as described in Pesaran et al. (2001) <doi:10.1002/jae.616> and provides the multipliers and the cointegrating equation. The validity and the accuracy of this package have been verified by successfully replicating the results of Pesaran et al. (2001) in Natsiopoulos and Tzeremes (2022) <doi:10.1002/jae.2919>.

Version 0.2.5

BugReports https://github.com/Natsiopoulos/ARDL/issues

License GPL-3

URL https://github.com/Natsiopoulos/ARDL

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Suggests strucchange, tseries, AICcmodavg, sandwich, testthat (>= 3.0.0)

Imports aod, dplyr, dynlm, gridExtra, ggplot2, lmtest, msm, stringr, zoo, stats

RoxygenNote 8.0.0

Config/testthat/edition 3

NeedsCompilation no

Author Kleanthis Natsiopoulos [aut, cre] (ORCID: <https://orcid.org/0000-0003-1180-2984>), Nickolaos Tzeremes [ths] (ORCID: <https://orcid.org/0000-0002-6938-3404>), Daniel Finnan [aut] (ORCID: <https://orcid.org/0009-0009-7468-5288>)

Maintainer Kleanthis Natsiopoulos <klnatsio@gmail.com>

Repository CRAN

Date/Publication 2026-05-10 05:12:54 UTC

Contents

`ardl` *ARDL model regression*

Description

A simple way to construct complex ARDL specifications providing just the model order additional to the model formula. It uses `dynlm` under the hood. `ardl` is a generic function and the default method constructs an 'ardl' model while the other method takes a model of `class` 'uecm' and converts in into an 'ardl'.

Usage

```
ardl(...)

## S3 method for class 'uecm'
ardl(object, ...)

## Default S3 method:
ardl(formula, data, order, start = NULL, end = NULL, ...)
```

Arguments

`...`	Additional arguments to be passed to the low level regression fitting functions.
`object`	An object of `class` 'uecm'.
`formula`	A "formula" describing the linear model. Details for model specification are given under 'Details'.

data	A time series object (e.g., "ts", "zoo" or "zooreg") or a data frame containing the variables in the model. In the case of a data frame, it is coerced into a ts object with start = 1, end = nrow(data) and frequency = 1. If not found in data, the variables are NOT taken from any environment.
order	A specification of the order of the ARDL model. A numeric vector of the same length as the total number of variables (excluding the fixed ones, see 'Details'). It should only contain positive integers or 0. An integer could be provided if all variables are of the same order.
start	Start of the time period which should be used for fitting the model.
end	End of the time period which should be used for fitting the model.

Details

The formula should contain only variables that exist in the data provided through data plus some additional functions supported by dynlm (i.e., trend()).

You can also specify fixed variables that are not supposed to be lagged (e.g. dummies etc.) simply by placing them after |. For example, y ~ x1 + x2 | z1 + z2 where z1 and z2 are the fixed variables and should not be considered in order. Note that the | notion should not be confused with the same notion in dynlm where it introduces instrumental variables.

Value

ardl returns an object of class c("ardl", "dynlm", "lm"). In addition, attributes 'order', 'data', 'parsed_formula' and 'full_formula' are provided.

Mathematical Formula

The general form of an $ARDL(p, q_1, \ldots, q_k)$ is:

$$y_t = c_0 + c_1 t + \sum_{i=1}^{p} b_{y,i} y_{t-i} + \sum_{j=1}^{k} \sum_{l=0}^{q_j} b_{j,l} x_{j,t-l} + \epsilon_t$$

Author(s)

Kleanthis Natsiopoulos, <klnatsio@gmail.com>

See Also

uecm, recm

Examples

```
data(denmark)

## Estimate an ARDL(3,1,3,2) model -------------------------------------

ardl_3132 <- ardl(LRM ~ LRY + IBO + IDE, data = denmark, order = c(3,1,3,2))
summary(ardl_3132)
```

```
## Add dummies or other variables that should stay fixed ---------------

d_74Q1_75Q3 <- ifelse(time(denmark) >= 1974 & time(denmark) <= 1975.5, 1, 0)

# the date can also be setted as below
d_74Q1_75Q3_ <- ifelse(time(denmark) >= "1974 Q1" & time(denmark) <= "1975 Q3", 1, 0)
identical(d_74Q1_75Q3, d_74Q1_75Q3_)
den <- cbind(denmark, d_74Q1_75Q3)
ardl_3132_d <- ardl(LRM ~ LRY + IBO + IDE | d_74Q1_75Q3,
                    data = den, order = c(3,1,3,2))
summary(ardl_3132_d)
compare <- data.frame(AIC = c(AIC(ardl_3132), AIC(ardl_3132_d)),
                      BIC = c(BIC(ardl_3132), BIC(ardl_3132_d)))
rownames(compare) <- c("no dummy", "with dummy")
compare

## Estimate an ARDL(3,1,3,2) model with a linear trend -----------------

ardl_3132_tr <- ardl(LRM ~ LRY + IBO + IDE + trend(LRM),
                     data = denmark, order = c(3,1,3,2))

# Alternative time trend specifications:
# time(LRM)                 1974 + (0, 1, ..., 55)/4 time(data)
# trend(LRM)                (1, 2, ..., 55)/4        (1:n)/freq
# trend(LRM, scale = FALSE) (1, 2, ..., 55)          1:n

## Subsample ARDL regression (start after 1975 Q4) ---------------------

ardl_3132_sub <- ardl(LRM ~ LRY + IBO + IDE, data = denmark,
                      order = c(3,1,3,2), start = "1975 Q4")

# the date can also be setted as below
ardl_3132_sub2 <- ardl(LRM ~ LRY + IBO + IDE, data = denmark,
                       order = c(3,1,3,2), start = c(1975,4))
identical(ardl_3132_sub, ardl_3132_sub2)
summary(ardl_3132_sub)

## Ease of use ----------------------------------------------------------

# The model specification of the ardl_3132 model can be created as easy as order=c(3,1,3,2)
# or else, it could be done using the dynlm package as:
library(dynlm)
m <- dynlm(LRM ~ L(LRM, 1) + L(LRM, 2) + L(LRM, 3) + LRY + L(LRY, 1) + IBO + L(IBO, 1) +
           L(IBO, 2) + L(IBO, 3) + IDE + L(IDE, 1) + L(IDE, 2), data = denmark)
identical(m$coefficients, ardl_3132$coefficients)

# The full formula can be extracted from the ARDL model, and this is equal to
ardl_3132$full_formula
m2 <- dynlm(ardl_3132$full_formula, data = ardl_3132$data)
identical(m$coefficients, m2$coefficients)

## Post-estimation testing ----------------------------------------------
```

```
# See examples in the help file of the uecm() function
```

`auto_ardl` *Automatic ARDL model selection*

Description

It searches for the best ARDL order specification, according to the selected criterion, taking into account the constraints provided.

Usage

```
auto_ardl(
  formula,
  data,
  max_order,
  fixed_order = -1,
  starting_order = NULL,
  selection = "AIC",
  selection_minmax = c("min", "max"),
  grid = FALSE,
  search_type = c("horizontal", "vertical"),
  start = NULL,
  end = NULL,
  ...
)
```

Arguments

`formula`	A "formula" describing the linear model. Details for model specification are given under 'Details' in the help file of the `ardl` function.
`data`	A time series object (e.g., "ts", "zoo" or "zooreg") or a data frame containing the variables in the model. In the case of a data frame, it is coerced into a `ts` object with `start = 1`, `end = nrow(data)` and `frequency = 1`. If not found in data, the variables are NOT taken from any environment.
`max_order`	It sets the maximum order for each variable where the search is taking place. A numeric vector of the same length as the total number of variables (excluding the fixed ones, see 'Details' in the help file of the `ardl` function). It should only contain positive integers. An integer could be provided if the maximum order for all variables is the same.
`fixed_order`	It allows setting a fixed order for some variables. The algorithm will not search for any other order than this. A numeric vector of the same length as the total number of variables (excluding the fixed ones). It should contain positive integers or 0 to set as a constraint. A -1 should be provided for any variable that should not be constrained. `fixed_order` overrides the corresponding `max_order` and `starting_order`.

`starting_order` Specifies the order for each variable from which each search will start. It is a numeric vector of the same length as the total number of variables (excluding the fixed ones). It should contain positive integers or 0 or only one integer could be provided if the starting order for all variables is the same. Default is set to NULL. If unspecified (`NULL`) and `grid = FALSE`, then all possible $ARDL(p)$ models are calculated (constraints are taken into account), where p is the minimum value in `max_order`. Note that where `starting_order` is provided, its first element will be the minimum value of p that the searching algorithm will consider (think of it like a 'minimum p order' restriction) (see 'Searching algorithm' below). If `grid = TRUE`, only the first argument (p) will have an effect.

`selection` A character string specifying the selection criterion according to which the candidate models will be ranked. Default is `AIC`. Any other selection criterion can be used (a user specified or a function from another package) as long as it can be applied as `selection(model)`. The preferred model is the one with the smaller value of the selection criterion. If the selection criterion works the other way around (the bigger the better), `selection_minmax = "max"` should also be supplied (see 'Examples' below).

`selection_minmax` A character string that indicates whether the criterion in `selection` is supposed to be minimized (default) or maximized.

`grid` If `FALSE` (default), the stepwise searching regression algorithm will search for the best model by adding and subtracting terms corresponding to different ARDL orders. If `TRUE`, the whole set of all possible ARDL models (accounting for constraints) will be evaluated. Note that this method can be very time-consuming in case that `max_order` is big and there are many independent variables that create a very big number of possible combinations.

`search_type` A character string describing the search type. If "horizontal" (default), the searching algorithm increases or decreases by 1 the order of each variable in each iteration. When the order of the last variable has been accessed, it begins again from the first variable until it converges. If "vertical", the searching algorithm increases or decreases by 1 the order of a variable until it converges. Then it continues the same for the next variable. The two options result to very similar top orders. The default ("horizontal"), sometimes is a little more accurate, but the "vertical" is almost 2 times faster. Not applicable if `grid = TRUE`.

`start` Start of the time period which should be used for fitting the model.

`end` End of the time period which should be used for fitting the model.

`...` Additional arguments to be passed to the low level regression fitting functions.

Value

`auto_ardl` returns a list which contains:

`best_model` An object of class `c("ardl", "dynlm", "lm")`

`best_order` A numeric vector with the order of the best model selected

`top_orders` A data.frame with the orders of the top 20 models

Searching algorithm

The algorithm performs the optimization process starting from multiple starting points concerning the autoregressive order p. The searching algorithm will perform a complete search, each time starting from a different starting order. These orders are presented in the tables below, for `grid = FALSE` and different values of `starting_order`.

`starting_order = NULL`:

ARDL(p)	->	p	q1	q2	...	qk
ARDL(1)	->	1	1	1	...	1
ARDL(2)	->	2	2	2	...	2
:	->	:	:	:	:	:
ARDL(P)	->	P	P	P	...	P

`starting_order = c(3, 0, 1, 2)`:

p	q1	q2	q3
3	0	1	2
4	0	1	2
:	:	:	:
P	0	1	2

Author(s)

Kleanthis Natsiopoulos, <klnatsio@gmail.com>

See Also

ardl

Examples

```
data(denmark)

## Find the best ARDL order -------------------------------------------

# Up to 5 for the autoregressive order (p) and 4 for the rest (q1, q2, q3)

# Using the defaults search_type = "horizontal", grid = FALSE and selection = "AIC"
# ("Not run" indications only for testing purposes)
## Not run:
model1 <- auto_ardl(LRM ~ LRY + IBO + IDE, data = denmark,
                    max_order = c(5,4,4,4))
model1$top_orders

## Same, with search_type = "vertical" --------------------------------

model1_h <- auto_ardl(LRM ~ LRY + IBO + IDE, data = denmark,
                      max_order = c(5,4,4,4), search_type = "vertical")
model1_h$top_orders
```

```
## Find the global optimum ARDL order ---------------------------------

# It may take more than 10 seconds
model_grid <- auto_ardl(LRM ~ LRY + IBO + IDE, data = denmark,
                        max_order = c(5,4,4,4), grid = TRUE)

## Different selection criteria ----------------------------------------

# Using BIC as selection criterion instead of AIC
model1_b <- auto_ardl(LRM ~ LRY + IBO + IDE, data = denmark,
                      max_order = c(5,4,4,4), selection = "BIC")
model1_b$top_orders

# Using other criteria like adjusted R squared (the bigger the better)
adjr2 <- function(x) { summary(x)$adj.r.squared }
model1_adjr2 <- auto_ardl(LRM ~ LRY + IBO + IDE, data = denmark,
                          max_order = c(5,4,4,4), selection = "adjr2",
                          selection_minmax = "max")
model1_adjr2$top_orders

# Using functions from other packages as selection criteria
if (requireNamespace("AICcmodavg", quietly = TRUE)) {

library(AICcmodavg)
model1_aicc <- auto_ardl(LRM ~ LRY + IBO + IDE, data = denmark,
                         max_order = c(5,4,4,4), selection = "AICc")
model1_aicc$top_orders
adjr2 <- function(x){ summary(x)$adj.r.squared }
model1_adjr2 <- auto_ardl(LRM ~ LRY + IBO + IDE, data = denmark,
                          max_order = c(5,4,4,4), selection = "adjr2",
                          selection_minmax = "max")
model1_adjr2$top_orders

## DIfferent starting order -------------------------------------------

# The searching algorithm will start from the following starting orders:
# p q1 q2 q3
# 1 1  3  2
# 2 1  3  2
# 3 1  3  2
# 4 1  3  2
# 5 1  3  2

model1_so <- auto_ardl(LRM ~ LRY + IBO + IDE, data = denmark,
                       max_order = c(5,4,4,4), starting_order = c(1,1,3,2))

# Starting from p=3 (don't search for p=1 and p=2)
# Starting orders:
# p q1 q2 q3
# 3 1  3  2
# 4 1  3  2
# 5 1  3  2
```

```
model1_so_3 <- auto_ardl(LRM ~ LRY + IBO + IDE, data = denmark,
                         max_order = c(5,4,4,4), starting_order = c(3,1,3,2))

# If starting_order = NULL, the starting orders for each iteration will be:
# p q1 q2 q3
# 1 1  1  1
# 2 2  2  2
# 3 3  3  3
# 4 4  4  4
# 5 5  5  5
}

## Add constraints ------------------------------------------------------

# Restrict only the order of IBO to be 2
model1_ibo2 <- auto_ardl(LRM ~ LRY + IBO + IDE, data = denmark,
                         max_order = c(5,4,4,4), fixed_order = c(-1,-1,2,-1))
model1_ibo2$top_orders

# Restrict the order of LRM to be 3 and the order of IBO to be 2
model1_lrm3_ibo2 <- auto_ardl(LRM ~ LRY + IBO + IDE, data = denmark,
                         max_order = c(5,4,4,4), fixed_order = c(3,-1,2,-1))
model1_lrm3_ibo2$top_orders

## Set the starting date for the regression (data starts at "1974 Q1") -

# Set regression starting date to "1976 Q1"
model1_76q1 <- auto_ardl(LRM ~ LRY + IBO + IDE, data = denmark,
                         max_order = c(5,4,4,4), start = "1976 Q1")
start(model1_76q1$best_model)

## End(Not run)
```

`bounds_f_test` *Bounds Wald-test for no cointegration*

Description

`bounds_f_test` performs the Wald bounds-test for no cointegration *Pesaran et al. (2001).* It is a Wald test on the parameters of a UECM (Unrestricted Error Correction Model) expressed either as a Chisq-statistic or as an F-statistic.

Usage

```
bounds_f_test(
  object,
  case,
  alpha = NULL,
  pvalue = TRUE,
  exact = FALSE,
```

```
  R = 40000,
  test = c("F", "Chisq"),
  vcov_matrix = NULL
)
```

Arguments

object	An object of class 'ardl' or 'uecm'.
case	An integer from 1-5 or a character string specifying whether the 'intercept' and/or the 'trend' have to participate in the short-run or the long-run relationship (cointegrating equation) (see section 'Cases' below).
alpha	A numeric value between 0 and 1 indicating the significance level of the critical value bounds. If NULL (default), no critical value bounds for a specific level of significance are provide, only the p-value. See section 'alpha, bounds and p-value' below for details.
pvalue	A logical indicating whether you want the p-value to be provided. The default is TRUE. See section 'alpha, bounds and p-value' below for details.
exact	A logical indicating whether you want asymptotic (T = 1000) or exact sample size critical value bounds and p-value. The default is FALSE for asymptotic. See section 'alpha, bounds and p-value' below for details.
R	An integer indicating how many iterations will be used if exact = TRUE. Default is 40000.
test	A character vector indicating whether you want the Wald test to be expressed as 'F' or as 'Chisq' statistic. Default is "F".
vcov_matrix	The estimated covariance matrix of the parameter estimates of the uecm model. Used to estimate the test statistic. The default is vcov(object) (when vcov_matrix = NULL), but other estimations of the covariance matrix of the regression's estimated coefficients can also be used (e.g., using vcovHC or vcovHAC). Only applicable if the input object is of class "uecm".

Value

A list with class "htest" containing the following components:

method	a character string indicating what type of test was performed.
alternative	a character string describing the alternative hypothesis.
statistic	the value of the test statistic.
null.value	the value of the population parameters k (the number of independent variables) and T (the number of observations) specified by the null hypothesis.
data.name	a character string giving the name(s) of the data.
parameters	numeric vector containing the critical value bounds.
p.value	the p-value of the test.
PSS2001parameters	numeric vector containing the critical value bounds as presented by *Pesaran et al. (2001)*. See section 'alpha, bounds and p-value' below for details.
tab	data.frame containing the statistic, the critical value bounds, the alpha level of significance and the p-value.

Hypothesis testing

$$\Delta y_t = c_0 + c_1 t + \pi_y y_{t-1} + \sum_{j=1}^{k} \pi_j x_{j,t-1} + \sum_{i=1}^{p-1} \psi_{y,i} \Delta y_{t-i} + \sum_{j=1}^{k} \sum_{l=1}^{q_j - 1} \psi_{j,l} \Delta x_{j,t-l} + \sum_{j=1}^{k} \omega_j \Delta x_{j,t} + \epsilon_t$$

Cases 1, 3, 5:

$$\mathbf{H_0}: \pi_y = \pi_1 = \cdots = \pi_k = 0$$

$$\mathbf{H_1}: \pi_y \neq \pi_1 \neq \ldots \neq \pi_k \neq 0$$

Case 2:

$$\mathbf{H_0}: \pi_y = \pi_1 = \cdots = \pi_k = c_0 = 0$$

$$\mathbf{H_1}: \pi_y \neq \pi_1 \neq \ldots \neq \pi_k \neq c_0 \neq 0$$

Case 4:

$$\mathbf{H_0}: \pi_y = \pi_1 = \cdots = \pi_k = c_1 = 0$$

$$\mathbf{H_1}: \pi_y \neq \pi_1 \neq \ldots \neq \pi_k \neq c_1 \neq 0$$

alpha, bounds and p-value

In this section it is explained how the critical value bounds and p-values are obtained.

- If `exact = FALSE`, then the asymptotic (T = 1000) critical value bounds and p-value are provided.
- Only the asymptotic critical value bounds and p-values, and only for k <= 10 are precalculated, everything else has to be computed.
- Precalculated critical value bounds and p-values were simulated using `set.seed(2020)` and `R = 70000`.
- Precalculated critical value bounds exist only for `alpha` being one of the 0.005, 0.01, 0.025, 0.05, 0.075, 0.1, 0.15 or 0.2, everything else has to be computed.
- If `alpha` is one of the 0.1, 0.05, 0.025 or 0.01 (and `exact = FALSE` and k <= 10), `PSS2001parameters` shows the critical value bounds presented in *Pesaran et al. (2001)* (less precise).

Cases

According to *Pesaran et al. (2001)*, we distinguish the long-run relationship (cointegrating equation) (and thus the bounds-test and the Restricted ECMs) between 5 different cases. These differ in terms of whether the 'intercept' and/or the 'trend' are restricted to participate in the long-run relationship or they are unrestricted and so they participate in the short-run relationship.

Case 1:
- No *intercept* and no *trend*.
- `case` inputs: 1 or "n" where "n" stands for none.

Case 2:
- Restricted *intercept* and no *trend*.
- `case` inputs: 2 or "rc" where "rc" stands for restricted constant.

Case 3:
- Unrestricted *intercept* and no *trend*.
- `case` inputs: 3 or "uc" where "uc" stands for unrestricted constant.

Case 4:
- Unrestricted *intercept* and restricted *trend*.
- `case` inputs: 4 or "ucrt" where "ucrt" stands for unrestricted constant and restricted trend.

Case 5:
- Unrestricted *intercept* and unrestricted *trend*.
- `case` inputs: 5 or "ucut" where "ucut" stands for unrestricted constant and unrestricted trend.

Note that you can't restrict (or leave unrestricted) a parameter that doesn't exist in the input model. For example, you can't compute `recm(object,case=3)` if the object is an ARDL (or UECM) model with no intercept. The same way, you can't compute `bounds_f_test(object, case=5)` if the object is an ARDL (or UECM) model with no linear trend.

References

Pesaran, M. H., Shin, Y., & Smith, R. J. (2001). Bounds testing approaches to the analysis of level relationships. *Journal of Applied Econometrics*, 16(3), 289-326

Author(s)

Kleanthis Natsiopoulos, <klnatsio@gmail.com>

See Also

`bounds_t_test` `ardl` `uecm`

Examples

```
data(denmark)

## How to use cases under different models (regarding deterministic terms)

## Construct an ARDL(3,1,3,2) model with different deterministic terms -

# Without constant
ardl_3132_n <- ardl(LRM ~ LRY + IBO + IDE -1, data = denmark, order = c(3,1,3,2))

# With constant
```

```
ardl_3132_c <- ardl(LRM ~ LRY + IBO + IDE, data = denmark, order = c(3,1,3,2))

# With constant and trend
ardl_3132_ct <- ardl(LRM ~ LRY + IBO + IDE + trend(LRM), data = denmark, order = c(3,1,3,2))

## F-bounds test for no level relationship (no cointegration) ----------

# For the model without a constant
bounds_f_test(ardl_3132_n, case = 1)
# or
bounds_f_test(ardl_3132_n, case = "n")

# For the model with a constant
# Including the constant term in the long-run relationship (restricted constant)
bounds_f_test(ardl_3132_c, case = 2)
# or
bounds_f_test(ardl_3132_c, case = "rc")

# Including the constant term in the short-run relationship (unrestricted constant)
bounds_f_test(ardl_3132_c, case = "uc")
# or
bounds_f_test(ardl_3132_c, case = 3)

# For the model with constant and trend
# Including the constant term in the short-run and the trend in the long-run relationship
# (unrestricted constant and restricted trend)
bounds_f_test(ardl_3132_ct, case = "ucrt")
# or
bounds_f_test(ardl_3132_ct, case = 4)

# For the model with constant and trend
# Including the constant term and the trend in the short-run relationship
# (unrestricted constant and unrestricted trend)
bounds_f_test(ardl_3132_ct, case = "ucut")
# or
bounds_f_test(ardl_3132_ct, case = 5)

## Note that you can't restrict a deterministic term that doesn't exist

# For example, the following tests will produce an error:
## Not run:
bounds_f_test(ardl_3132_c, case = 1)
bounds_f_test(ardl_3132_ct, case = 3)
bounds_f_test(ardl_3132_c, case = 4)

## End(Not run)

## Asymptotic p-value and critical value bounds (assuming T = 1000) ----

# Include critical value bounds for a certain level of significance

# F-statistic is larger than the I(1) bound (for a=0.05) as expected (p-value < 0.05)
bft <- bounds_f_test(ardl_3132_c, case = 2, alpha = 0.05)
```

```
bft
bft$tab

# Traditional but less precise critical value bounds, as presented in Pesaran et al. (2001)
bft$PSS2001parameters

# F-statistic is slightly larger than the I(1) bound (for a=0.005)
# as p-value is slightly smaller than 0.005
bounds_f_test(ardl_3132_c, case = 2, alpha = 0.005)

## Exact sample size p-value and critical value bounds -----------------

# Setting a seed is suggested to allow the replication of results
# 'R' can be increased for more accurate resutls

# F-statistic is smaller than the I(1) bound (for a=0.01) as expected (p-value > 0.01)
# Note that the exact sample p-value (0.01285) is very different than the
# asymptotic (0.004418)
# It can take more than 30 seconds
## Not run:
set.seed(2020)
bounds_f_test(ardl_3132_c, case = 2, alpha = 0.01, exact = TRUE)

## End(Not run)

## "F" and "Chisq" statistics ----------------------------------------

# The p-value is the same, the test-statistic and critical value bounds are different
# but analogous
bounds_f_test(ardl_3132_c, case = 2, alpha = 0.01)
bounds_f_test(ardl_3132_c, case = 2, alpha = 0.01, test = "Chisq")
```

---

bounds_t_test *Bounds t-test for no cointegration*

Description

bounds_t_test performs the t-bounds test for no cointegration *Pesaran et al. (2001).* It is a t-test on the parameters of a UECM (Unrestricted Error Correction Model).

Usage

```
bounds_t_test(
  object,
  case,
  alpha = NULL,
  pvalue = TRUE,
  exact = FALSE,
  R = 40000,
  vcov_matrix = NULL
)
```

Arguments

object	An object of class 'ardl' or 'uecm'.
case	An integer (1, 3 or 5) or a character string specifying whether the 'intercept' and/or the 'trend' have to participate in the short-run relationship (see section 'Cases' below). Note that the t-bounds test can't be applied for cases 2 and 4.
alpha	A numeric value between 0 and 1 indicating the significance level of the critical value bounds. If NULL (default), no critical value bounds for a specific level of significance are provide, only the p-value. See section 'alpha, bounds and p-value' below for details.
pvalue	A logical indicating whether you want the p-value to be provided. The default is TRUE. See section 'alpha, bounds and p-value' below for details.
exact	A logical indicating whether you want asymptotic (T = 1000) or exact sample size critical value bounds and p-value. The default is FALSE for asymptotic. See section 'alpha, bounds and p-value' below for details.
R	An integer indicating how many iterations will be used if exact = TRUE. Default is 40000.
vcov_matrix	The estimated covariance matrix of the parameter estimates of the uecm model. Used to estimate the test statistic. The default is vcov(object) (when vcov_matrix = NULL), but other estimations of the covariance matrix of the regression's estimated coefficients can also be used (e.g., using vcovHC or vcovHAC). Only applicable if the input object is of class "uecm".

Value

A list with class "htest" containing the following components:

method	a character string indicating what type of test was performed.
alternative	a character string describing the alternative hypothesis.
statistic	the value of the test statistic.
null.value	the value of the population parameters k (the number of independent variables) and T (the number of observations) specified by the null hypothesis.
data.name	a character string giving the name(s) of the data.
parameters	numeric vector containing the critical value bounds.
p.value	the p-value of the test.
PSS2001parameters	numeric vector containing the critical value bounds as presented by *Pesaran et al. (2001)*. See section 'alpha, bounds and p-value' below for details.
tab	data.frame containing the statistic, the critical value bounds, the alpha level of significance and the p-value.

Hypothesis testing

$$\Delta y_t = c_0 + c_1 t + \pi_y y_{t-1} + \sum_{j=1}^{k} \pi_j x_{j,t-1} + \sum_{i=1}^{p-1} \psi_{y,i} \Delta y_{t-i} + \sum_{j=1}^{k} \sum_{l=1}^{q_j - 1} \psi_{j,l} \Delta x_{j,t-l} + \sum_{j=1}^{k} \omega_j \Delta x_{j,t} + \epsilon_t$$

$$\mathbf{H_0}: \pi_y = 0$$
$$\mathbf{H_1}: \pi_y \neq 0$$

alpha, bounds and p-value

In this section it is explained how the critical value bounds and p-values are obtained.

- If `exact = FALSE`, then the asymptotic (T = 1000) critical value bounds and p-value are provided.
- Only the asymptotic critical value bounds and p-values, and only for k <= 10 are precalculated, everything else has to be computed.
- Precalculated critical value bounds and p-values were simulated using `set.seed(2020)` and `R = 70000`.
- Precalculated critical value bounds exist only for `alpha` being one of the 0.005, 0.01, 0.025, 0.05, 0.075, 0.1, 0.15 or 0.2, everything else has to be computed.
- If `alpha` is one of the 0.1, 0.05, 0.025 or 0.01 (and `exact = FALSE` and k <= 10), `PSS2001parameters` shows the critical value bounds presented in *Pesaran et al. (2001)* (less precise).

Cases

According to *Pesaran et al. (2001)*, we distinguish the long-run relationship (cointegrating equation) (and thus the bounds-test and the Restricted ECMs) between 5 different cases. These differ in terms of whether the 'intercept' and/or the 'trend' are restricted to participate in the long-run relationship or they are unrestricted and so they participate in the short-run relationship.

Case 1:
- No *intercept* and no *trend*.
- `case` inputs: 1 or "n" where "n" stands for none.

Case 2:
- Restricted *intercept* and no *trend*.
- `case` inputs: 2 or "rc" where "rc" stands for restricted constant.

Case 3:
- Unrestricted *intercept* and no *trend*.
- `case` inputs: 3 or "uc" where "uc" stands for unrestricted constant.

Case 4:
- Unrestricted *intercept* and restricted *trend*.
- `case` inputs: 4 or "ucrt" where "ucrt" stands for unrestricted constant and restricted trend.

Case 5:
- Unrestricted *intercept* and unrestricted *trend*.
- `case` inputs: 5 or "ucut" where "ucut" stands for unrestricted constant and unrestricted trend.

Note that you can't restrict (or leave unrestricted) a parameter that doesn't exist in the input model. For example, you can't compute `recm(object,case=3)` if the object is an ARDL (or UECM) model with no intercept. The same way, you can't compute `bounds_f_test(object, case=5)` if the object is an ARDL (or UECM) model with no linear trend.

References

Pesaran, M. H., Shin, Y., & Smith, R. J. (2001). Bounds testing approaches to the analysis of level relationships. *Journal of Applied Econometrics*, 16(3), 289-326

Author(s)

Kleanthis Natsiopoulos, <klnatsio@gmail.com>

See Also

bounds_f_test ardl uecm

Examples

```
data(denmark)

## How to use cases under different models (regarding deterministic terms)

## Construct an ARDL(3,1,3,2) model with different deterministic terms -

# Without constant
ardl_3132_n <- ardl(LRM ~ LRY + IBO + IDE -1, data = denmark, order = c(3,1,3,2))

# With constant
ardl_3132_c <- ardl(LRM ~ LRY + IBO + IDE, data = denmark, order = c(3,1,3,2))

# With constant and trend
ardl_3132_ct <- ardl(LRM ~ LRY + IBO + IDE + trend(LRM), data = denmark, order = c(3,1,3,2))

## t-bounds test for no level relationship (no cointegration) ----------

# For the model without a constant
bounds_t_test(ardl_3132_n, case = 1)
# or
bounds_t_test(ardl_3132_n, case = "n")

# For the model with a constant
# Including the constant term in the short-run relationship (unrestricted constant)
bounds_t_test(ardl_3132_c, case = "uc")
# or
bounds_t_test(ardl_3132_c, case = 3)

# For the model with constant and trend
# Including the constant term and the trend in the short-run relationship
# (unrestricted constant and unrestricted trend)
bounds_t_test(ardl_3132_ct, case = "ucut")
# or
bounds_t_test(ardl_3132_ct, case = 5)

## Note that you can't use bounds t-test for cases 2 and 4, or use a wrong model

# For example, the following tests will produce an error:
## Not run:
bounds_t_test(ardl_3132_n, case = 2)
bounds_t_test(ardl_3132_c, case = 4)
bounds_t_test(ardl_3132_ct, case = 3)
```

```
## End(Not run)

## Asymptotic p-value and critical value bounds (assuming T = 1000) ----

# Include critical value bounds for a certain level of significance

# t-statistic is larger than the I(1) bound (for a=0.05) as expected (p-value < 0.05)
btt <- bounds_t_test(ardl_3132_c, case = 3, alpha = 0.05)
btt
btt$tab

# Traditional but less precise critical value bounds, as presented in Pesaran et al. (2001)
btt$PSS2001parameters

# t-statistic doesn't exceed the I(1) bound (for a=0.005) as p-value is greater than 0.005
bounds_t_test(ardl_3132_c, case = 3, alpha = 0.005)

## Exact sample size p-value and critical value bounds -----------------

# Setting a seed is suggested to allow the replication of results
# 'R' can be increased for more accurate resutls

# t-statistic is smaller than the I(1) bound (for a=0.01) as expected (p-value > 0.01)
# Note that the exact sample p-value (0.009874) is very different than the
# asymptotic (0.005538)
# It can take more than 90 seconds
## Not run:
set.seed(2020)
bounds_t_test(ardl_3132_c, case = 3, alpha = 0.01, exact = TRUE)

## End(Not run)
```

`coint_eq` *Cointegrating equation (long-run level relationship)*

Description

Creates the cointegrating equation (long-run level relationship) providing an 'ardl', 'uecm' or 'recm' model.

Usage

```
coint_eq(object, case)

## S3 method for class 'recm'
coint_eq(object, ...)

## Default S3 method:
coint_eq(object, case)
```

Arguments

`object`	An object of `class` 'ardl', 'uecm' or 'recm'.
`case`	An integer from 1-5 or a character string specifying whether the 'intercept' and/or the 'trend' have to participate in the long-run level relationship (cointegrating equation) (see section 'Cases' below). If the input object is of class 'recm', `case` is not needed as the model is already under a certain case.
`...`	Currently unused argument.

Value

`coint_eq` returns an numeric vector containing the cointegrating equation.

Cases

According to *Pesaran et al. (2001)*, we distinguish the long-run relationship (cointegrating equation) (and thus the bounds-test and the Restricted ECMs) between 5 different cases. These differ in terms of whether the 'intercept' and/or the 'trend' are restricted to participate in the long-run relationship or they are unrestricted and so they participate in the short-run relationship.

Case 1:
- No *intercept* and no *trend*.
- `case` inputs: 1 or "n" where "n" stands for none.

Case 2:
- Restricted *intercept* and no *trend*.
- `case` inputs: 2 or "rc" where "rc" stands for restricted constant.

Case 3:
- Unrestricted *intercept* and no *trend*.
- `case` inputs: 3 or "uc" where "uc" stands for unrestricted constant.

Case 4:
- Unrestricted *intercept* and restricted *trend*.
- `case` inputs: 4 or "ucrt" where "ucrt" stands for unrestricted constant and restricted trend.

Case 5:
- Unrestricted *intercept* and unrestricted *trend*.
- `case` inputs: 5 or "ucut" where "ucut" stands for unrestricted constant and unrestricted trend.

Note that you can't restrict (or leave unrestricted) a parameter that doesn't exist in the input model. For example, you can't compute `recm(object,case=3)` if the object is an ARDL (or UECM) model with no intercept. The same way, you can't compute `bounds_f_test(object, case=5)` if the object is an ARDL (or UECM) model with no linear trend.

References

Pesaran, M. H., Shin, Y., & Smith, R. J. (2001). Bounds testing approaches to the analysis of level relationships. *Journal of Applied Econometrics*, 16(3), 289-326

Author(s)

Kleanthis Natsiopoulos, <klnatsio@gmail.com>

See Also

`plot_lr` `ardl` `uecm` `recm` `bounds_f_test` `bounds_t_test`

Examples

```
data(denmark)
library(zoo) # for cbind.zoo()

## Estimate the Cointegrating Equation of an ARDL(3,1,3,2) model -------

# From an ARDL model (under case 2, restricted constant)
ardl_3132 <- ardl(LRM ~ LRY + IBO + IDE, data = denmark, order = c(3,1,3,2))
ce2_ardl <- coint_eq(ardl_3132, case = 2)

# From an UECM (under case 2, restricted constant)
uecm_3132 <- uecm(ardl_3132)
ce2_uecm <- coint_eq(uecm_3132, case = 2)

# From a RECM (under case 2, restricted constant)
# Notice that if a RECM has already been estimated under a certain case,
# the 'coint_eq()' can't be under different case, so no 'case' argument needed.
recm_3132 <- recm(uecm_3132, case = 2)
# The RECM is already under case 2, so the 'case' argument is no needed
ce2_recm <- coint_eq(recm_3132)

identical(ce2_ardl, ce2_uecm, ce2_recm)

## Check for a degenerate level relationship ---------------------------

# The bounds F-test under both cases reject the Null Hypothesis of no level relationship.
bounds_f_test(ardl_3132, case = 2)
bounds_f_test(ardl_3132, case = 3)

# The bounds t-test also rejects the NUll Hypothesis of no level relationship.
bounds_t_test(ardl_3132, case = 3)

# But when the constant enters the long-run equation (case 3)
# this becomes a degenerate relationship.
ce3_ardl <- coint_eq(ardl_3132, case = 3)

plot_lr(ardl_3132, coint_eq = ce2_ardl, show.legend = TRUE)

plot_lr(ardl_3132, coint_eq = ce3_ardl, show.legend = TRUE)
plot_lr(ardl_3132, coint_eq = ce3_ardl, facets = TRUE, show.legend = TRUE)
```

---

denmark *The Danish data on money income prices and interest rates*

Description

This data set contains the series used by S. Johansen and K. Juselius for estimating a money demand function of Denmark.

Usage

```
denmark
```

Format

A time-series object with 55 rows and 5 variables. Time period from 1974:Q1 until 1987:Q3.

LRM logarithm of real money, M2

LRY logarithm of real income

LPY logarithm of price deflator

IBO bond rate

IDE bank deposit rate

Details

An object of class "zooreg" "zoo".

Source

https://onlinelibrary.wiley.com/doi/10.1111/j.1468-0084.1990.mp52002003.x

References

Johansen, S. and Juselius, K. (1990), Maximum Likelihood Estimation and Inference on Cointegration – with Applications to the Demand for Money, *Oxford Bulletin of Economics and Statistics*, **52, 2**, 169–210.

---

multipliers *Multipliers estimation*

Description

`multipliers` is a generic function used to estimate short-run (impact), delay, interim and long-run (total) multipliers, accompanied by their corresponding standard errors, t-statistics and p-values.

Usage

```
multipliers(object, type = "lr", vcov_matrix = NULL, se = FALSE)

## S3 method for class 'ardl'
multipliers(object, type = "lr", vcov_matrix = NULL, se = FALSE)

## S3 method for class 'uecm'
multipliers(object, type = "lr", vcov_matrix = NULL, se = FALSE)
```

Arguments

object	An object of class 'ardl' or 'uecm'.
type	A character string describing the type of multipliers. Use "lr" for long-run (total) multipliers (default), "sr" or 0 for short-run (impact) multipliers or an integer between 1 and 200 for delay and interim multipliers.
vcov_matrix	The estimated covariance matrix of the parameter estimates. Used by the transformation function to estimate the standard errors (and so the t-statistics and p-values) of the multipliers. The default is vcov(object) (when vcov_matrix = NULL), but other estimations of the covariance matrix of the regression's estimated coefficients can also be used (e.g., using vcovHC or vcovHAC).
se	A logical indicating whether you want standard errors for delay multipliers to be provided. The default is FALSE. Note that this parameter does not refer to the standard errors for the long-run and short-run multipliers, for which are always calculated. IMPORTANT: Calculating standard errors for long periods of delays may cause your computer to run out of memory and terminate your R session, losing important unsaved work. As a rule of thumb, try not to exceed type = 19 when se = TRUE.

Details

The function invokes two different methods, one for objects of class 'ardl' and one for objects of class 'uecm'. This is because of the different (but equivalent) transformation functions that are used for each class/model ('ardl' and 'uecm') to estimate the multipliers.

type = 0 is equivalent to type = "sr".

Note that the interim multipliers are the cumulative sum of the delays, and that the sum of the delay multipliers (for long enough periods) and thus a distant enough interim multiplier match the long-run multipliers.

The delay (interim) multiplier can be interpreted as the effect on the dependent variable in period t+s, resulting from an instant (sustained) shock to an independent variable in period t.

The delta method is used for approximating the standard errors (and thus the t-statistics and p-values) of the estimated long-run and delay multipliers.

Value

multipliers returns (for long and short run multipliers) a data.frame containing the independent variables (including possibly existing intercept or trend and excluding the fixed variables) and their corresponding standard errors, t-statistics and p-values. For delay and interim multipliers it returns a list with a data.frame for each variable, containing the delay and interim multipliers for each period.

Mathematical Formula

Short-Run Multipliers:

As derived from an ARDL:

$$\frac{\partial y_t}{\partial x_{j,t}} = b_{j,0} \qquad j \in \{1, \ldots, k\}$$

As derived from an Unrestricted ECM:

$$\frac{\partial y_t}{\partial x_{j,t}} = \omega_j \quad j \in \{1, \ldots, k\}$$

Constant and Linear Trend:

$$c_0$$

$$c_1$$

Delay & Interim Multipliers:

As derived from an ARDL:

$$Delay_{x_j,s} = \frac{\partial y_{t+s}}{\partial x_{j,t}} = b_{j,s} + \sum_{i=1}^{min\{p,s\}} b_{y,i} \frac{\partial y_{t+(s-i)}}{\partial x_{j,t}} \quad b_{j,s} = 0 \ \forall \ s > q$$

$$Interim_{x_j,s} = \sum_{i=0}^{s} Delay_{x_j,s}$$

Constant and Linear Trend:

$$Delay_{intercept,s} = c_0 + \sum_{i=1}^{min\{p,s\}} b_{y,i} Delay_{intercept,s-i} \quad c_0 = 0 \ \forall \ s \neq 0$$

$$Interim_{intercept,s} = \sum_{i=0}^{s} Delay_{intercept,s}$$

$$Delay_{trend,s} = c_1 + \sum_{i=1}^{min\{p,s\}} b_{y,i} Delay_{trend,s-i} \quad c_1 = 0 \ \forall \ s \neq 0$$

$$Interim_{trend,s} = \sum_{i=0}^{s} Delay_{trend,s}$$

Long-Run Multipliers:

As derived from an ARDL:

$$\frac{\partial y_{t+\infty}}{\partial x_{j,t}} = \theta_j = \frac{\sum_{l=0}^{q_j} b_{j,l}}{1 - \sum_{i=1}^{p} b_{y,i}} \quad j \in \{1, \ldots, k\}$$

Constant and Linear Trend:

$$\mu = \frac{c_0}{1 - \sum_{i=1}^{p} b_{y,i}}$$

$$\delta = \frac{c_1}{1 - \sum_{i=1}^{p} b_{y,i}}$$

As derived from an Unrestricted ECM:

$$\frac{\partial y_{t+\infty}}{\partial x_{j,t}} = \theta_j = \frac{\pi_j}{-\pi_y} \qquad j \in \{1, \ldots, k\}$$

Constant and Linear Trend:

$$\mu = \frac{c_0}{-\pi_y}$$

$$\delta = \frac{c_1}{-\pi_y}$$

Author(s)

Kleanthis Natsiopoulos, <klnatsio@gmail.com>

See Also

ardl, uecm, plot_delay

Examples

```
data(denmark)

## Estimate the long-run multipliers of an ARDL(3,1,3,2) model ---------

# From an ARDL model
ardl_3132 <- ardl(LRM ~ LRY + IBO + IDE, data = denmark, order = c(3,1,3,2))
mult_ardl <- multipliers(ardl_3132)
mult_ardl

# From an UECM
uecm_3132 <- uecm(ardl_3132)
mult_uecm <- multipliers(uecm_3132)
mult_uecm

all.equal(mult_ardl, mult_uecm)


## Estimate the short-run multipliers of an ARDL(3,1,3,2) model --------

mult_sr <- multipliers(uecm_3132, type = "sr")
mult_0 <- multipliers(uecm_3132, type = 0)
all.equal(mult_sr, mult_0)
```

```
## Estimate the delay & interim multipliers of an ARDL(3,1,3,2) model --

mult_lr <- multipliers(uecm_3132, type = "lr")
mult_inter80 <- multipliers(uecm_3132, type = 80)

mult_lr
sum(mult_inter80$`(Intercept)`$Delay)
mult_inter80$`(Intercept)`$Interim[nrow(mult_inter80$`(Intercept)`)]
sum(mult_inter80$LRY$Delay)
mult_inter80$LRY$Interim[nrow(mult_inter80$LRY)]
sum(mult_inter80$IBO$Delay)
mult_inter80$IBO$Interim[nrow(mult_inter80$IBO)]
sum(mult_inter80$IDE$Delay)
mult_inter80$IDE$Interim[nrow(mult_inter80$IDE)]
plot(mult_inter80$LRY$Delay, type='l')
plot(mult_inter80$LRY$Interim, type='l')

mult_inter12 <- multipliers(uecm_3132, type = 12, se = TRUE)
plot_delay(mult_inter12, interval = 0.95)
```

NT2022 *The UK earnings equation data from Natsiopoulos and Tzeremes (2022)*

Description

This data set contains the series used by Natsiopoulos and Tzeremes (2022) for re-estimating the UK earnings equation. The clean format of the data retrieved from the Data Archive of Natsiopoulos and Tzeremes (2022).

Usage

```
NT2022
```

Format

A time-series object with 196 rows and 9 variables. Time period from 1971:Q1 until 2019:Q4.

time time variable

w real wage

Prod labor productivity

UR unemployment rate

Wedge wedge effect

Union union power

D7475 income policies 1974:Q1-1975:Q4

D7579 income policies 1975:Q1-1979:Q4

UnionR union membership

Details

An object of class "zooreg" "zoo".

Source

http://qed.econ.queensu.ca/jae/datasets/natsiopoulos001/

References

Kleanthis Natsiopoulos and Nickolaos G. Tzeremes, (2022), "ARDL bounds test for Cointegration: Replicating the Pesaran et al. (2001) Results for the UK Earnings Equation Using R", *Journal of Applied Econometrics*, **37, 5**, 1079–1090. doi:10.1002/jae.2919

`plot_delay` *Create plots for the delay multipliers*

Description

Creates plots for the delay multipliers and their uncertainty intervals based on their estimated standard errors. This is a basic `ggplot` with a few customizable parameters.

Usage

```
plot_delay(
  multipliers,
  facets_ncol = 2,
  interval = FALSE,
  interval_color = "blue",
  show.legend = FALSE,
  xlab = "Period",
  ylab = "Delay",
  ...
)
```

Arguments

`multipliers`	A list returned from `multipliers`, in which `type` is a positive integer to return delay multipliers.
`facets_ncol`	If a positive integer, it indicates the number of the columns in the facet. If FALSE, each plot is created separately. The default is 2.
`interval`	If FALSE (default), no uncertainty intervals are drawn. If a positive integer, the intervals are this number times the standard error. If a number between 0 and 1 (e.g. 0.95), the equivalent confidence interval is drawn (e.g. 95% CI). In case of the confidence intervals, they are based on the Gaussian distribution.
`interval_color`	The color of the uncertainty intervals. Default is "blue".
`show.legend`	A logical indicating whether the interval legend is shown. Default is FALSE.

xlab, ylab	Names displayed at the x and y axes respectively. Default is "Period" and "Delay" respectively.
...	Currently unused argument.

Value

plot_delay returns a number of ggplot objects.

Author(s)

Kleanthis Natsiopoulos, <klnatsio@gmail.com>

See Also

multipliers

Examples

```
ardl_3132 <- ardl(LRM ~ LRY + IBO + IDE, data = denmark, order = c(3,1,3,2))
delay_mult <- multipliers(ardl_3132, type = 12, se = TRUE)

## Simply plot the delay multipliers ----------------------------------

plot_delay(delay_mult)

## Rearrange them -----------------------------------------------------

plot_delay(delay_mult, facets_ncol = 1)

## Add 1 standard deviation uncertainty intervals ----------------------

plot_delay(delay_mult, interval = 1)

## Add 95% confidence intervals, change color and add legend -----------

plot_delay(delay_mult, interval = 0.95, interval_color = "darkgrey",
           show.legend = TRUE)
```

---

plot_lr *Create plot for the long-run (cointegrating) equation*

Description

Creates a plot for the long-run relationship in comparison with the dependent variable, and the fitted values of the model. This is a basic ggplot with a few customizable parameters.

Usage

```
plot_lr(
  object,
  coint_eq,
  facets = FALSE,
  show_fitted = FALSE,
  show.legend = FALSE,
  xlab = "Time",
  ...
)
```

Arguments

object	An object of class 'ardl'.
coint_eq	The objected returned from coint_eq.
facets	A logical indicating whether the long-run relationship appears in a separate plot. Default is FALSE.
show_fitted	A logical indicating whether the fitted values are shown. Default is FALSE.
show.legend	A logical indicating whether the legend is shown. Default is FALSE.
xlab	Name displayed at the x axis. Default is "Time".
...	Currently unused argument.

Value

plot_lr returns a ggplot object.

Author(s)

Kleanthis Natsiopoulos, <klnatsio@gmail.com>

See Also

coint_eq

Examples

```
ardl_3132 <- ardl(LRM ~ LRY + IBO + IDE, data = denmark, order = c(3,1,3,2))
ce2 <- coint_eq(ardl_3132, case = 2)

plot_lr(ardl_3132, coint_eq = ce2)

## Compare fitted values and place long-run relationship separately ----

ce3 <- coint_eq(ardl_3132, case = 3)
plot_lr(ardl_3132, coint_eq = ce3, facets = TRUE, show_fitted = TRUE,
        show.legend = TRUE)
```

`predict.ardl` *Predict method for ARDL model fits*

Description

Makes predictions based on fitted ARDL models, extending the generic function `predict` for use with `ardl`. It is designed to take advantage of lags of the dependent variable (AR components) from the fitted model and new values of the independent variable(s) to calculate predictions.

Usage

```
## S3 method for class 'ardl'
predict(object, newdata, ...)
```

Arguments

`object`	A fitted model of `class ardl`.
`newdata`	A time series object (`ts`, `zoo`, `zooreg`) or a `data.frame` containing the new values of the independent variables x required to make predictions.
`...`	further arguments passed to or from other methods.

Details

`predict.ardl` works recursively to calculate predictions based on $y_{t-1}, ..., y_{t-p}$ values contained within a fitted model, and new values of $x_{T+1}, ..., x_{T+h}$, where T denotes the last observation of t from the original data and h is the number of new observations that will be used to calculate predictions. These are supplied through the `newdata` argument. It is important to note that for a given prediction $\hat{y}_{T+1}$, the independent variables provided through `newdata`, are expected to follow consecutively from the last observation x_T used to create the model, i.e. $x_{T+1}, ..., x_{T+h}$, with no gaps. If a time series object is passed to the function, the frequency and whether the observations follow consecutively will be checked.

Value

`predict.ardl` returns values for $\hat{y}_{T+1}, ..., \hat{y}_{T+h}$ equal in length to the number of rows in `newdata`, as a `vector`.

Calculation example

We want to predict $\hat{y}_{T+1}$ given an $ARDL(2, 2)$ model:

$$\hat{y}_{T+1} = Z\boldsymbol{\beta}$$

where Z is our design matrix and β is a vector of our model coefficients. We need 2 lags for our dependent and independent variables: y_T, y_{T-1} and X_T, X_{T-1}. Plus, X_{T+1} (from `newdata`), which in our design matrix is equivalent to time t in our original model:

$$Z = \left[\begin{array}{c|cccccc} & \mathbf{c} & \mathbf{y_{t-1}} & \mathbf{y_{t-2}} & \mathbf{x_t} & \mathbf{x_{t-1}} & \mathbf{x_{t-2}} \\ \hline T{+}1 & 1 & y_T & y_{T-1} & x_{T+1} & x_T & x_{T-1} \end{array}\right]$$

We proceed to the next iteration, predicting $\hat{y}_{T+2}$, using the previously calculated value of $\hat{y}_{T+1}$:

$$Z = \left[\begin{array}{c|cccccc} & \mathbf{c} & \mathbf{y_{t-1}} & \mathbf{y_{t-2}} & \mathbf{x_t} & \mathbf{x_{t-1}} & \mathbf{x_{t-2}} \\ \hline T+2 & 1 & y_{T+1} & y_T & x_{T+2} & x_{T+1} & x_T \end{array}\right]$$

Iteration continues until we've calculated all the necessary values of $\hat{y}_{T+h}$ for `newdata`.

Author(s)

Kleanthis Natsiopoulos, <klnatsio@gmail.com>

Daniel Finnan, <dan@custom-made.org.uk>

See Also

ardl

Examples

```
data(denmark)
## Estimate an ARDL(2,1,2,1,1) model, using a subset of the denmark data
ardl_21211 <- ardl(LRM ~ LRY + LPY + IBO + IDE, data = denmark,
                   order = c(2,1,2,1,1),
                   start = "1974 Q1", end = "1986 Q2")
## Make predictions based on the remainder of the denmark data not used in
## training the model
predict_data <- window(denmark, start = "1986 Q3", end = "1987 Q3")
## Drop the dependent variable, since we're predicting it
predict_data <- predict_data[, c("LRY", "LPY", "IBO", "IDE")]
predictions <- predict(ardl_21211, predict_data)

## Estimate a ARDL(4,4,4,4) model with a linear trend and dummies
## Create dummies
d_74Q1_75Q3 <- ifelse(time(denmark) >= "1974 Q1" & time(denmark) <= "1975 Q3", 1, 0)
# Add them to the data
denmark_dums <- cbind(denmark, d_74Q1_75Q3)
## Estimate the model
ardl_4444 <- ardl(LRM ~ LRY + LPY + IBO + trend(LRM) | d_74Q1_75Q3,
                  data = denmark_dums,
                  order = c(4,4,4,4),
                  start = "1974 Q1", end = "1986 Q2")
## Take the remaining data not used in training the model
predict_data <- window(denmark_dums, start = "1986 Q3", end = "1987 Q3")
## Drop the dependent variable
predict_data <- predict_data[, c("LRY", "LPY", "IBO", "d_74Q1_75Q3")]
## Compute the predictions
predictions <- predict(ardl_4444, predict_data)
```

PSS2001 *The UK earnings equation data from Pesaran et al. (2001)*

Description

This data set contains the series used by Pesaran et al. (2001) for estimating the UK earnings equation. The clean format of the data retrieved from the Data Archive of Natsiopoulos and Tzeremes (2022).

Usage

```
PSS2001
```

Format

A time-series object with 112 rows and 7 variables. Time period from 1970:Q1 until 1997:Q4.

w real wage

Prod labor productivity

UR unemployment rate

Wedge wedge effect

Union union power

D7475 income policies 1974:Q1-1975:Q4

D7579 income policies 1975:Q1-1979:Q4

Details

An object of class "zooreg" "zoo".

Source

http://qed.econ.queensu.ca/jae/datasets/pesaran001/
http://qed.econ.queensu.ca/jae/datasets/natsiopoulos001/

References

M. Hashem Pesaran, Richard J. Smith, and Yongcheol Shin, (2001), "Bounds Testing Approaches to the Analysis of Level Relationships", *Journal of Applied Econometrics*, **16, 3**, 289–326.

Kleanthis Natsiopoulos and Nickolaos G. Tzeremes, (2022), "ARDL bounds test for Cointegration: Replicating the Pesaran et al. (2001) Results for the UK Earnings Equation Using R", *Journal of Applied Econometrics*, **37, 5**, 1079–1090. doi:10.1002/jae.2919

`recm` *Restricted ECM regression*

Description

Creates the Restricted Error Correction Model (RECM). This is the conditional RECM, which is the RECM of the underlying ARDL.

Usage

```
recm(object, case)
```

Arguments

`object`	An object of `class` 'ardl' or 'uecm'.
`case`	An integer from 1-5 or a character string specifying whether the 'intercept' and/or the 'trend' have to participate in the short-run or the long-run relationship (cointegrating equation) (see section 'Cases' below).

Details

Note that the statistical significance of 'ect' in a RECM should not be tested using the corresponding t-statistic (or the p-value) because it doesn't follow a standard t-distribution. Instead, the `bounds_t_test` should be used.

Value

`recm` returns an object of `class` `c("recm", "dynlm", "lm")`. In addition, attributes 'order', 'data', 'parsed_formula' and 'full_formula' are provided.

Mathematical Formula

The formula of a Restricted ECM conditional to an $ARDL(p, q_1, \dots, q_k)$ is:

$$\Delta y_t = c_0 + c_1 t + \sum_{i=1}^{p-1} \psi_{y,i} \Delta y_{t-i} + \sum_{j=1}^{k} \sum_{l=1}^{q_j - 1} \psi_{j,l} \Delta x_{j,t-l} + \sum_{j=1}^{k} \omega_j \Delta x_{j,t} + \pi_y ECT_t + \epsilon_t$$

$$\psi_{j,l} = 0 \ \forall \ q_j = 1, \psi_{j,l} = \omega_j = 0 \ \forall \ q_j = 0$$

Under Case 1:
- $c_0 = c_1 = 0$
- $ECT = y_{t-1} - (\sum_{j=1}^{k} \theta_j x_{j,t-1})$

Under Case 2:
- $c_0 = c_1 = 0$
- $ECT = y_{t-1} - (\mu + \sum_{j=1}^{k} \theta_j x_{j,t-1})$

Under Case 3:
- $c_1 = 0$
- $ECT = y_{t-1} - (\sum_{j=1}^{k} \theta_j x_{j,t-1})$

Under Case 4:
- $c_1 = 0$

- $ECT = y_{t-1} - (\delta(t-1) + \sum_{j=1}^{k} \theta_j x_{j,t-1})$

Under Case 5: • $ECT = y_{t-1} - (\sum_{j=1}^{k} \theta_j x_{j,t-1})$

In all cases, $x_{j,t-1}$ in ECT is replaced by $x_{j,t} \quad \forall \; q_j = 0$

Cases

According to *Pesaran et al. (2001)*, we distinguish the long-run relationship (cointegrating equation) (and thus the bounds-test and the Restricted ECMs) between 5 different cases. These differ in terms of whether the 'intercept' and/or the 'trend' are restricted to participate in the long-run relationship or they are unrestricted and so they participate in the short-run relationship.

Case 1:
- No *intercept* and no *trend*.
- `case` inputs: 1 or "n" where "n" stands for none.

Case 2:
- Restricted *intercept* and no *trend*.
- `case` inputs: 2 or "rc" where "rc" stands for restricted constant.

Case 3:
- Unrestricted *intercept* and no *trend*.
- `case` inputs: 3 or "uc" where "uc" stands for unrestricted constant.

Case 4:
- Unrestricted *intercept* and restricted *trend*.
- `case` inputs: 4 or "ucrt" where "ucrt" stands for unrestricted constant and restricted trend.

Case 5:
- Unrestricted *intercept* and unrestricted *trend*.
- `case` inputs: 5 or "ucut" where "ucut" stands for unrestricted constant and unrestricted trend.

Note that you can't restrict (or leave unrestricted) a parameter that doesn't exist in the input model. For example, you can't compute `recm(object,case=3)` if the object is an ARDL (or UECM) model with no intercept. The same way, you can't compute `bounds_f_test(object, case=5)` if the object is an ARDL (or UECM) model with no linear trend.

References

Pesaran, M. H., Shin, Y., & Smith, R. J. (2001). Bounds testing approaches to the analysis of level relationships. *Journal of Applied Econometrics*, 16(3), 289-326

Author(s)

Kleanthis Natsiopoulos, <klnatsio@gmail.com>

See Also

`ardl` `uecm`

Examples

```
data(denmark)

## Estimate the RECM, conditional to it's underlying ARDL(3,1,3,2) -----

# Indirectly from an ARDL
ardl_3132 <- ardl(LRM ~ LRY + IBO + IDE, data = denmark, order = c(3,1,3,2))
recm_3132 <- recm(ardl_3132, case = 2)

# Indirectly from an UECM
uecm_3132 <- uecm(ardl_3132)
recm_3132_ <- recm(uecm_3132, case = 2)
identical(recm_3132, recm_3132_)
summary(recm_3132)

## Error Correction Term (ect) & Speed of Adjustment -------------------

# The coefficient of the ect,
# shows the Speed of Adjustment towards equilibrium.
# Note that this can be also be obtained from an UECM,
# through the coefficient of the term L(y, 1) (where y is the dependent variable).
tail(recm_3132$coefficients, 1)
uecm_3132$coefficients[2]

## Post-estimation testing ---------------------------------------------

# See examples in the help file of the uecm() function
```

---

to_lm *Convert dynlm model (ardl, uecm, recm) to lm model*

Description

Takes a dynlm model of class 'ardl', 'uecm' or 'recm' and converts it into an lm model. This can help using the model as a regular lm model with functions that are not compatible with dynlm models such as the predict function to forecast.

Usage

```
to_lm(object, fix_names = FALSE, data_class = NULL, ...)
```

Arguments

object	An object of class 'ardl', 'uecm' or 'recm'.
fix_names	A logical, indicating whether the variable names should be rewritten without special functions and character in the names such as "d()" or "L()". When fix_names = TRUE, the characters "(", and "," are replaces with ".", and ")" and spaces are deleted. The name of the dependent variable is always transformed, regardless of the value of this parameter. Default is FALSE.

data_class	If "ts", it converts the data class to ts (see examples for its usage). The default is NULL, which uses the same data provided in the original object.
...	Currently unused argument.

Value

to_lm returns an object of class "lm".

Author(s)

Kleanthis Natsiopoulos, <klnatsio@gmail.com>

See Also

ardl, uecm, recm

Examples

```
## Convert ARDL into lm -----------------------------------------------

ardl_3132 <- ardl(LRM ~ LRY + IBO + IDE, data = denmark, order = c(3,1,3,2))
ardl_3132_lm <- to_lm(ardl_3132)
summary(ardl_3132)$coefficients
summary(ardl_3132_lm)$coefficients

## Convert UECM into lm -----------------------------------------------

uecm_3132 <- uecm(ardl_3132)
uecm_3132_lm <- to_lm(uecm_3132)
summary(uecm_3132)$coefficients
summary(uecm_3132_lm)$coefficients

## Convert RECM into lm -----------------------------------------------

recm_3132 <- recm(ardl_3132, case = 2)
recm_3132_lm <- to_lm(recm_3132)
summary(recm_3132)$coefficients
summary(recm_3132_lm)$coefficients

## Use the lm model to forecast ---------------------------------------

# Forecast using the in-sample data
insample_data <- ardl_3132$model
head(insample_data)
predicted_values <- predict(ardl_3132_lm, newdata = insample_data)

# The predicted values are expected to be the same as the fitted values
ardl_3132$fitted.values
predicted_values

# Convert to ts class for the plot
predicted_values <- ts(predicted_values, start = c(1974,4), frequency=4)
```

```
plot(denmark$LRM, lwd=4) #The input dependent variable
lines(ardl_3132$fitted.values, lwd=4, col="blue") #The fitted values
lines(predicted_values, lty=2, lwd=2, col="red") #The predicted values

## Convert to lm for post-estimation testing ---------------------------

# Ramsey's RESET test for functional form
library(lmtest) # for resettest()
library(strucchange) # for efp(), and sctest()

## Not run:
    # This produces an error.
    # resettest() cannot use data of class 'zoo' such as the 'denmark' data
    # used to build the original model
    resettest(uecm_3132, type = c("regressor"))

## End(Not run)

uecm_3132_lm <- to_lm(uecm_3132, data_class = "ts")
resettest(uecm_3132_lm, power = 2)

# CUSUM test for structural change detection
## Not run:
    # This produces an error.
    # efp() does not understand special functions such as "d()" and "L()"
    efp(uecm_3132$full_formula, data = uecm_3132$model)

## End(Not run)

uecm_3132_lm_names <- to_lm(uecm_3132, fix_names = TRUE)
fluctuation <- efp(uecm_3132_lm_names$full_formula,
                   data = uecm_3132_lm_names$model)
sctest(fluctuation)
plot(fluctuation)
```

---

uecm *Unrestricted ECM regression*

Description

uecm is a generic function used to construct Unrestricted Error Correction Models (UECM). The function invokes two different methods. The default method works exactly like ardl. The other method requires an object of class 'ardl'. Both methods create the conditional UECM, which is the UECM of the underlying ARDL.

Usage

```
uecm(...)
```

```
## S3 method for class 'ardl'
uecm(object, ...)

## Default S3 method:
uecm(formula, data, order, start = NULL, end = NULL, ...)
```

Arguments

`...`	Additional arguments to be passed to the low level regression fitting functions.
`object`	An object of `class` 'ardl'.
`formula`	A "formula" describing the linear model. Details for model specification are given under 'Details'.
`data`	A time series object (e.g., "ts", "zoo" or "zooreg") or a data frame containing the variables in the model. In the case of a data frame, it is coerced into a `ts` object with `start = 1`, `end = nrow(data)` and `frequency = 1`. If not found in data, the variables are NOT taken from any environment.
`order`	A specification of the order of the underlying ARDL model (e.g., for the UECM of an ARDL(1,0,2) model it should be `order = c(1,0,2)`). A numeric vector of the same length as the total number of variables (excluding the fixed ones, see 'Details'). It should only contain positive integers or 0. An integer could be provided if all variables are of the same order.
`start`	Start of the time period which should be used for fitting the model.
`end`	End of the time period which should be used for fitting the model.

Details

The `formula` should contain only variables that exist in the data provided through `data` plus some additional functions supported by `dynlm` (i.e., `trend()`).

You can also specify fixed variables that are not supposed to be lagged (e.g. dummies etc.) simply by placing them after `|`. For example, `y ~ x1 + x2 | z1 + z2` where `z1` and `z2` are the fixed variables and should not be considered in `order`. Note that the `|` notion should not be confused with the same notion in `dynlm` where it introduces instrumental variables.

Value

`uecm` returns an object of `class` `c("uecm", "dynlm", "lm")`. In addition, attributes 'order', 'data', 'parsed_formula' and 'full_formula' are provided.

Mathematical Formula

The formula of an Unrestricted ECM conditional to an $ARDL(p, q_1, \dots, q_k)$ is:

$$\Delta y_t = c_0 + c_1 t + \pi_y y_{t-1} + \sum_{j=1}^{k} \pi_j x_{j,t-1} + \sum_{i=1}^{p-1} \psi_{y,i} \Delta y_{t-i} + \sum_{j=1}^{k} \sum_{l=1}^{q_j - 1} \psi_{j,l} \Delta x_{j,t-l} + \sum_{j=1}^{k} \omega_j \Delta x_{j,t} + \epsilon_t$$

$$\psi_{j,l} = 0 \ \ \forall \ \ q_j \leq 1, \qquad \psi_{y,i} = 0 \ \ if \ \ p = 1$$

In addition, $x_{j,t-1}$ and $\Delta x_{j,t}$ cancel out becoming $x_{j,t} \ \ \forall \ \ q_j = 0$

Author(s)

Kleanthis Natsiopoulos, <klnatsio@gmail.com>

See Also

ardl recm

Examples

```
data(denmark)

## Estimate the UECM, conditional to it's underlying ARDL(3,1,3,2) -----

# Indirectly
ardl_3132 <- ardl(LRM ~ LRY + IBO + IDE, data = denmark, order = c(3,1,3,2))
uecm_3132 <- uecm(ardl_3132)

# Directly
uecm_3132_ <- uecm(LRM ~ LRY + IBO + IDE, data = denmark, order = c(3,1,3,2))
identical(uecm_3132, uecm_3132_)
summary(uecm_3132)

## Post-estimation testing ------------------------------------------------

library(lmtest) # for bgtest(), bptest(), and resettest()
library(tseries) # for jarque.bera.test()
library(strucchange) # for efp(), and sctest()

# Breusch-Godfrey test for higher-order serial correlation
bgtest(uecm_3132, order = 4)

# Breusch-Pagan test against heteroskedasticity
bptest(uecm_3132)

# Ramsey's RESET test for functional form
## Not run:
    # This produces an error.
    # resettest() cannot use data of class 'zoo' such as the 'denmark' data
    # used to build the original model
    resettest(uecm_3132, type = c("regressor"))

## End(Not run)

uecm_3132_lm <- to_lm(uecm_3132, data_class = "ts")
resettest(uecm_3132_lm, power = 2)

# Jarque-Bera test for normality
jarque.bera.test(residuals(uecm_3132))

# CUSUM test for structural change detection
## Not run:
    # This produces an error.
```

```
    # efp() does not understand special functions such as "d()" and "L()"
    efp(uecm_3132$full_formula, data = uecm_3132$model)

## End(Not run)

uecm_3132_lm_names <- to_lm(uecm_3132, fix_names = TRUE)
fluctuation <- efp(uecm_3132_lm_names$full_formula,
                   data = uecm_3132_lm_names$model)
sctest(fluctuation)
plot(fluctuation)
```

Index