Package: kardl (via r-universe)

May 11, 2026

Type Package

Title Make Symmetric and Asymmetric ARDL Estimations

Version 1.3.1

Maintainer Huseyin Karamelikli <hakperest@gmail.com>

Description Implements estimation procedures for Autoregressive Distributed Lag (ARDL) and Nonlinear ARDL (NARDL) models, which allow researchers to investigate both short- and long-run relationships in time series data under mixed orders of integration. The package supports simultaneous modeling of symmetric and asymmetric regressors, flexible treatment of short-run and long-run asymmetries, and automated equation handling. It includes several cointegration testing approaches such as the Pesaran-Shin-Smith F and t bounds tests, and narayan test. Methodological foundations are provided in Pesaran, Shin, and Smith (2001) <doi:10.1016/S0304-4076(01)00049-5> and Shin, Yu, and Greenwood-Nimmo (2014, ISBN:9780123855079).

License GPL-3

Encoding UTF-8

LazyData true

Imports stats, msm, lmtest, nlWaldTest, car, ggplot2, utils

RoxygenNote 7.3.3

Suggests knitr, rmarkdown, officer, flextable, equatags, magrittr, rlang, tidyr, dplyr, MASS, testthat (>= 3.0.0)

NeedsCompilation no

VignetteBuilder knitr

Author Huseyin Karamelikli [aut, cre] (ORCID: <https://orcid.org/0000-0001-7622-0972>), Huseyin Utku Demir [aut] (ORCID: <https://orcid.org/0000-0002-9140-0362>)

Depends R (>= 3.5.0)

Config/testthat/edition 3

URL https://karamelikli.github.io/kardl/,
https://github.com/karamelikli/kardl

BugReports https://github.com/karamelikli/kardl/issues

Roxygen list (markdown = TRUE, roclets = c (``namespace'', ``rd'', ``srr::srr_stats_roclet''))

Config/pak/sysreqs cmake make libicu-dev

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

Date/Publication 2026-05-11 06:51:09 UTC

RemoteUrl https://github.com/karamelikli/kardl

RemoteRef HEAD

RemoteSha ab45a71128bd8b9458906726ef1e42a4a1afd10e

Contents

`bootstrap` *Bootstrap Confidence Intervals for Dynamic Multipliers*

Description

This function computes bootstrap confidence intervals (CI) for dynamic multipliers of a specified variable in a model estimated using the `kardl` package. The bootstrap method generates resampled datasets to estimate the variability of the dynamic multipliers, providing upper and lower bounds for the confidence interval.

Usage

```
bootstrap(
  kmodel,
  horizon = 80,
  replications = 100,
  level = 95,
  minProb = 0,
  seed = NULL
)
```

Arguments

kmodel	The model produced by the kardl function. This is the model object from which the dynamic multipliers are calculated.
horizon	An integer specifying the horizon over which dynamic multipliers will be computed. The horizon defines the time frame for the analysis (e.g., 40 periods).
replications	An integer indicating the number of bootstrap replications to perform. Higher values increase accuracy but also computational time. Default is 100.
level	A numeric value specifying the confidence level for the intervals (e.g., 95 for 95% confidence). Default is 90.
minProb	A numeric value specifying the minimum p-value threshold for including coefficients in the bootstrap. Coefficients with p-values above this threshold will be set to zero in the bootstrap samples. Default is 0 (no threshold). This parameter allows users to control the inclusion of coefficients in the bootstrap process based on their statistical significance. Setting a threshold can help focus the analysis on more relevant variables, but it may also exclude potentially important effects if set too stringently.
seed	An optional integer to set the random seed for reproducibility of the bootstrap results. If not provided, the bootstrap will use the current random state.

Details

The mpsi component of the output contains the dynamic multiplier estimates along with their upper and lower confidence intervals. These values are provided for each variable and at each time horizon.

Value

A list containing the following elements:

- **mpsi**: A data frame containing the dynamic multiplier estimates along with their upper and lower confidence intervals for each variable and time horizon.
- **level**: The confidence level used for the intervals (e.g 95).
- **horizon**: The horizon over which the multipliers were computed (e.g., 40).
- **vars**: A list of variable information extracted from the model, including dependent variable, independent variables, asymmetric variables, and deterministic terms.

- **replications**: The number of bootstrap replications performed.
- **type**: A character string indicating the type of analysis, in this case "bootstrap".

See Also

`mplier` for calculating dynamic multipliers

Examples

```
 # Example usage of the bootstrap function


# Fit a model using kardl
kardl_model <- kardl(imf_example_data,
                     CPI ~ ER + PPI + asy(ER) +
                      det(covid) + trend,
                     mode = c(1, 2, 3, 0))

# Perform bootstrap with specific variables for plotting
boot <-
  bootstrap(kardl_model,   replications=5, seed = 123L)
# The boot object will include all plots for the specified variables
# Displaying the boot object provides an overview of its components
names(boot)


# Inspect the first few rows of the dynamic multiplier estimates
 head(boot$mpsi)


 summary(boot)

# Retrieve plots generated during the bootstrap process
# Accessing all plots
 plot(boot)

# Accessing the plot for a specific variable by its name
 plot(boot, variable = "PPI")
plot(boot, variable = "ER")




library(magrittr)

   imf_example_data %>% kardl( CPI ~ PPI + asym(ER) +trend, maxlag=2) %>%
   bootstrap(replications=5) %>% plot(variable = "ER")
```

ecm *Estimate a Restricted ECM Model*

Description

The ecm function estimates a restricted Error Correction Model (ECM) based on the provided data and model specification. This function is designed to test for cointegration using the PSS t Bound test, which assesses the presence of a long-term equilibrium relationship between the dependent variable and the independent variables in the model.

Usage

```
ecm(
  data = NULL,
  formula = NULL,
  maxlag = NULL,
  mode = NULL,
  criterion = NULL,
  differentAsymLag = NULL,
  batch = NULL,
  ...
)
```

Arguments

data	The data of analysis
formula	A formula specifying the long-run model equation. This formula defines the relationships between the dependent variable and explanatory variables, including options for deterministic terms, asymmetric variables, and a trend component.

```
 Example formula:
\code{y ~ x + z + Asymmetric(z) + Lasymmetric(x2 + x3) + Sasymmetric(x3 + x4) + determi
```

Details

The formula allows flexible specification of variables and their roles:

- **Deterministic variables:** Deterministic regressors (e.g., dummy variables) can be included using `deterministic()`. Multiple deterministic variables may be supplied using `+`, for example `deterministic(dummy1 + dummy2)`. These variables are treated as fixed components and are not associated with short-run or long-run dynamics.
- **Asymmetric variables:** Asymmetric decompositions can be specified for short-run and/or long-run dynamics:
  - **Sasymmetric**: Specifies short-run asymmetric variables. For example, `Sasymmetric(x1 + x2)` applies short-run asymmetric decomposition to `x1` and `x2`.

- **Lasymmetric**: Specifies long-run asymmetric variables. For example, `Lasymmetric(x1 + x3)` applies long-run asymmetric decomposition to `x1` and `x3`.
- **Asymmetric**: Specifies variables that enter both short-run and long-run asymmetric decompositions. For example, `Asymmetric(x1 + x3)` applies asymmetric decomposition in both dynamics.

A **trend** term may be included to capture deterministic linear time trends by simply adding `trend` to the formula.

The formula also supports the use of `.` to represent all available regressors in the supplied data (excluding the dependent variable), following standard R formula conventions.

All of the operators `Deterministic()`, `Sasymmetric()`, `Lasymmetric()`, and `Asymmetric()` follow the same usage rules:

- They can be freely combined within a single formula, for example:

```
y ~ . +
  Asymmetric(z) +
  Lasymmetric(x2 + x3) +
  Sasymmetric(x3 + x4) +
  deterministic(dummy1 + dummy2) +
  trend
```

- They must not be nested within one another. Valid usage: `y ~ x + deterministic(dummy) + Asymmetric(z)`. Invalid usage (to be avoided): `y ~ x + deterministic(Asymmetric(z))` or `y ~ x + Asymmetric(deterministic(dummy))`.
- Where applicable, arguments are validated internally using `match.arg()`. Consequently, abbreviated inputs are accepted provided they uniquely identify a valid option. For example, if `"asymmetric"` is an admissible value, specifying `"a"` is sufficient. For clarity and reproducibility, however, full argument names are recommended.

These components may therefore be combined flexibly to construct a specification tailored to the empirical analysis.

`maxlag` An integer specifying the maximum number of lags to be considered for the model. The default value is `4`. This parameter sets an upper limit on the lag length during the model estimation process.

details

The `maxlag` parameter is crucial for defining the maximum lag length that the model will evaluate when selecting the optimal lag structure based on the specified `criterion`. It controls the computational effort and helps prevent overfitting by restricting the search space for lag selection.

- If the data has a short time horizon or is prone to overfitting, consider reducing `maxlag`.
- If the data is expected to have long-term dependencies, increasing `maxlag` may be necessary to capture the relevant dynamics.

Setting an appropriate value for `maxlag` depends on the nature of your dataset and the context of the analysis:

- For small datasets or quick tests, use smaller values (e.g., `maxlag = 2`).
- For datasets with more observations or longer-term patterns, larger values (e.g., `maxlag = 8`) may be appropriate, though this increases computational time.

examples

Using the default maximum lag (4)

```
kardl(data, MyFormula, maxlag = 4)
```

Reducing the maximum lag to 2 for faster computation

```
kardl(data, MyFormula, maxlag = 2)
```

Increasing the maximum lag to 8 for datasets with longer dependencies

```
kardl(data, MyFormula, maxlag = 8)
```

`mode` Specifies the mode of estimation and output control. This parameter determines how the function handles lag estimation and what kind of feedback or control is provided during the process. The available options are:

- **"quick"** (default): Displays progress and messages in the console while the function estimates the optimal lag values. This mode is suitable for interactive use or for users who want to monitor the estimation process in real-time. It provides detailed feedback for debugging or observation but may use additional resources due to verbose output.
- **"grid"** : Displays progress and messages in the console while the function estimates the optimal lag values. This mode is suitable for interactive use or for users who want to monitor the estimation process in real-time. It provides detailed feedback for debugging or observation but may use additional resources due to verbose output.
- **"grid_custom"**: Suppresses most or all console output, prioritizing faster execution and reduced resource usage on PCs or servers. This mode is recommended for high-performance scenarios, batch processing, or when the estimation process does not require user monitoring. Suitable for large-scale or repeated runs where output is unnecessary.
- **User-defined vector**: A numeric vector of lag values specified by the user, allowing full customization of the lag structure used in model estimation. When a user-defined vector is provided (e.g., `c(1, 2, 4, 5)`), the function skips the lag optimization process and directly uses the specified lags.
  Users can define lag values directly as a numeric vector. For example: `mode = c(1, 2, 4, 5)` assigns lags of 1, 2, 4, and 5 to variables in the specified order. Alternatively, lag values can be assigned to variables by name for clarity and control. For example: `mode = c(CPI = 2, ER_POS = 3, ER_NEG = 1, PPI = 3)` assigns lags to variables explicitly. Ensure that the lags are correctly designated by verifying the result using `kardl_model$properLag` after estimation.
  Attention! A function-based criterion or user-defined function can be specified for model selection, but this is only supported for `mode = "grid_custom"` and `mode = "quick"`. The `mode = "grid"` option is restricted to predefined criteria (e.g., AIC or BIC). For more information on available criteria, see the `modelCriterion` function documentation.
  - When using a numeric vector, ensure the order of lag values matches the variables in your formula.

  - If using named vectors, double-check the variable names to avoid mismatches or unintended results.
  - This mode bypasses the automatic lag optimization and assumes the user-defined lags are correct.

`criterion` A string specifying the information criterion to be used for selecting the optimal lag structure. The available options are:

- **"AIC"**: Akaike Information Criterion (default). This criterion balances model fit and complexity, favoring models that explain the data well with fewer parameters.
- **"BIC"**: Bayesian Information Criterion. This criterion imposes a stronger penalty for model complexity than AIC, making it more conservative in selecting models with fewer parameters.
- **"AICc"**: Corrected Akaike Information Criterion. This is an adjusted version of AIC that accounts for small sample sizes, making it more suitable when the number of observations is limited relative to the number of parameters.
- **"HQ"**: Hannan-Quinn Information Criterion. This criterion provides a compromise between AIC and BIC, favoring models that balance fit and complexity without being overly conservative.

The criterion can be specified as a string (e.g., `"AIC"`) or as a user-defined function that takes a fitted model object. Please visit the `modelCriterion` function documentation for more details on using custom criteria.

`differentAsymLag` A logical value indicating whether to allow different lag lengths for positive and negative decompositions.

`batch` A string specifying the batch processing configuration in the format "current_batch/total_batches". If a user utilize grid or grid_custom mode and want to split the lag search into multiple batches, this parameter can be used to define the current batch and the total number of batches. For example, "2/5" indicates that the current batch is the second out of a total of five batches. The default value is "1/1", meaning that the entire lag search is performed in a single batch.

`...` Additional arguments that can be passed to the function. These arguments can be used to specify other settings or parameters that are not explicitly defined in the main arguments.

Value

A list containing the results of the restricted ECM test, including:

- `ecm`: The estimated ECM model objects including:
  - `longrunEQ`: The estimated long-run model equation object.
  - `shortrunEQ`: The estimated short-run model equation
  - `ecmL`: The estimated long-run model object.
- **argsInfo**: A list of input arguments used for the estimation. It includes the data, formula, maxlag, mode, criterion, differentAsymLag, and batch settings.

- **extractedInfo**: A list containing extracted information from the input data and formula, such as variable names, deterministic terms, asymmetric variables, and the prepared dataset for estimation.
- **timeInfo**: A list containing timing information for the estimation process, including start time, end time, and total duration.
- **lagInfo**: A list containing lag selection information, including the optimal lag configuration and criteria values for different lag combinations.
- **estInfo**: A list containing estimation details, such as the type of model, estimation method, model formula, number of parameters (k), number of observations (n), start and end points of the fitted values, and total time span.
- **model**: The fitted linear model object of class `lm` representing the estimated ARDL or NARDL model.

Hypothesis testing

The null and alternative hypotheses for the restricted ECM test are as follows:

$$\mathbf{H_0}: \theta = 0$$

$$\mathbf{H_1}: \theta \neq 0$$

The null hypothesis (H_0) states that there is no cointegration in the model, meaning that the long-run relationship between the variables is not significant. The alternative hypothesis (H_1) suggests that there is cointegration, indicating a significant long-term relationship between the variables.

The test statistic is calculated as the t-statistic of the coefficient of the error correction term (θ) in the ECM model. If the absolute value of the t-statistic exceeds the critical value from the PSS t Bound table, we reject the null hypothesis in favor of the alternative hypothesis, indicating that cointegration is present.

The cases for the restricted ECM Bound test are defined as follows:

- `case 1`: No constant, no trend.

  This case is used when the model does not include a constant term or a trend term. It is suitable for models where the variables are stationary and do not exhibit any long-term trends.

  The model is specified as follows:

$$\Delta y_t = \sum_{j=1}^{p} \gamma_j \Delta y_{t-j} + \sum_{i=1}^{k} \sum_{j=0}^{q_i} \beta_{ij} \Delta x_{i,t-j} + \theta(y_{t-1} - \sum_{i=1}^{k} \alpha_i x_{i,t-1}) + e_t$$

- `case 2`: Restricted constant, no trend.

  This case is used when the model includes a constant term but no trend term. It is suitable for models where the variables exhibit a long-term relationship but do not have a trend component. The model is specified as follows:

$$\Delta y_t = \sum_{j=1}^{p} \gamma_j \Delta y_{t-j} + \sum_{i=1}^{k} \sum_{j=0}^{q_i} \beta_{ij} \Delta x_{i,t-j} + \theta(y_{t-1} - \alpha_0 - \sum_{i=1}^{k} \alpha_i x_{i,t-1}) + e_t$$

- `case 3`: Unrestricted constant, no trend.

  This case is used when the model includes an unrestricted constant term but no trend term. It is suitable for models where the variables exhibit a long-term relationship with a constant but do not have a trend component.

  The model is specified as follows:

$$\Delta y_t = \sum_{j=1}^{p} \gamma_j \Delta y_{t-j} + \sum_{i=1}^{k} \sum_{j=0}^{q_i} \beta_{ij} \Delta x_{i,t-j} + \theta(y_{t-1} - \alpha_0 - \sum_{i=1}^{k} \alpha_i x_{i,t-1}) + e_t$$

- `case 4`: Unrestricted Constant, restricted trend.

  This case is used when the model includes an unrestricted constant term and a restricted trend term. It is suitable for models where the variables exhibit a long-term relationship with a constant and a trend component.

  The model is specified as follows:

$$\Delta y_t = \phi + \sum_{j=1}^{p} \gamma_j \Delta y_{t-j} + \sum_{i=1}^{k} \sum_{j=0}^{q_i} \beta_{ij} \Delta x_{i,t-j} + \theta(y_{t-1} - \pi(t-1) - \sum_{i=1}^{k} \alpha_i x_{i,t-1}) + e_t$$

- `case 5`: Unrestricted constant, unrestricted trend.

The Error Correction Model (ECM) is specified as follows:

$$\Delta y_t = \phi + \varphi t + \sum_{j=1}^{p} \gamma_j \Delta y_{t-j} + \sum_{i=1}^{k} \sum_{j=0}^{q_i} \beta_{ij} \Delta x_{i,t-j} + \theta(y_{t-1} - \sum_{i=1}^{k} \alpha_i x_{i,t-1}) + e_t$$

Notation of reported coefficients

In the reported coefficients, the prefix `L` denotes lagged variables, where the accompanying number indicates the lag order, and `.d.` denotes first differences. Accordingly, `L1.PPI` represents the first lag of the level of `PPI` (long-run component), while `L3.d.PPI` denotes the third lag of the first-differenced `PPI` (short-run component).

In addition, the suffixes `_POS` and `_NEG` indicate the positive and negative partial sum components of a variable, respectively. This notation is used by default and remains valid unless modified through the `kardl_set()` function.

See Also

`kardl` `pssf` `psst` `ecm` `narayan`

Examples

```
# Sample article: THE DYNAMICS OF EXCHANGE RATE PASS-THROUGH TO DOMESTIC PRICES IN TURKEY
kardl_set(
  formula = CPI ~ ER + PPI + asym(ER) + deterministic(covid) + trend,
  data = imf_example_data,
  maxlag = 3
)
```

```
# Using the grid mode with batch processing to decrease execution time
ecm_model_grid <- ecm(mode = "grid")
ecm_model_grid

# Checking the cointegration test results using Pesaran t test
psst(ecm_model_grid)

# Getting the details of psst result
summary(psst(ecm_model_grid))

# Using the grid_custom mode for faster execution without console output
ecm_model <- ecm(imf_example_data, mode = "grid_custom", criterion = "HQ", batch = "2/3")
ecm_model

# Estimating the model with user-defined lag values
ecm_model2 <- ecm(mode = c(2, 1, 1, 3))

# Getting the results
ecm_model2

# Getting the summary of the results
summary(ecm_model2)

# Alternative specification
summary(ecm(imf_example_data, CPI ~ PPI + asym(ER) + trend, case = 4))

# For increasing the performance of finding the most fitted lag vector
ecm(mode = "grid_custom")

# Setting max lag instead of default value [4]
ecm(maxlag = 2, mode = "grid_custom")

# Using another criterion for finding the best lag
ecm(criterion = "HQ", mode = "grid_custom")

# For using different lag values for positive and negative decompositions
# Setting the same lags for positive and negative decompositions
kardl_set(differentAsymLag = FALSE)

diffAsymLags <- ecm(mode = "grid_custom")
diffAsymLags$lagInfo$OptLag

# Setting different lags for positive and negative decompositions
sameAsymLags <- ecm(differentAsymLag = TRUE, mode = "grid_custom")
sameAsymLags$lagInfo$OptLag

# Setting the prefixes and suffixes for nonlinear variables
kardl_reset()
kardl_set(AsymPrefix = c("asyP_", "asyN_"), AsymSuffix = c("_PP", "_NN"))
customizedNames <- ecm(imf_example_data, CPI ~ ER + PPI + asym(ER))
customizedNames
```

`imf_example_data` *IMF Example Data*

Description

This is an example data set used for testing purposes. It contains monthly data on exchange rates, consumer price index, producer price index, and a dummy variable for the COVID-19 pandemic for Turkey from January 1985 to February 2024.

Usage

```
imf_example_data
```

Format

A data frame with 470 rows and 4 variables:

ER Numeric. Exchange rate of Turkey.

CPI Numeric. CPI of Turkey.

PPI Numeric. PPI of Turkey.

covid Integer.Covid19 dummy variable.

Details

These data obtained from **imf.data** package. The sample data is not updated and obtained by following codes:

```
install.packages("imf.data")
library("imf.data")
IFS <- load_datasets("IFS")
```

- **PCPI_IX** Prices, Consumer Price Index, All items, Index
- **AIP_IX** Economic Activity, Industrial Production, Index
- **ENDE_XDC_USD_RATE** Exchange Rates, Domestic Currency per U.S. Dollar, End of Period, Rate

```
 trdata<-IFS$get_series(freq = "M", ref_area = "TR", indicator = c("PCPI_IX","AIP_IX","ENDE_XDC_USD_RATE
= "1985-01",end_period = "2024-02")
PeriodRow<-trdata[,1]
trdata[,1]<-NULL
colnames(trdata)<-c("ER","CPI","PPI")
trdata<-log(as.data.frame(lapply(trdata, function(x) as.numeric(x))))
rownames(trdata)<-PeriodRow
```

Inserting covid dummy variable

```
 trdata<-cbind(trdata,covid=0)
trdata[420:470,4]<-1
```

See Also

load_datasets

Examples

```
data(imf_example_data)
head(imf_example_data)
```

---

kardl *Estimate ARDL and NARDL Models with Automatic Lag Selection*

Description

This function estimates an Autoregressive Distributed Lag (ARDL) or Nonlinear ARDL (NARDL) model based on the provided data and model formula. It allows for flexible specification of variables, including deterministic terms, asymmetric variables, and trend components. The function also supports automatic lag selection using various information criteria.

Usage

```
kardl(
  data = NULL,
  formula = NULL,
  maxlag = NULL,
  mode = NULL,
  criterion = NULL,
  differentAsymLag = NULL,
  batch = NULL,
  ...
)
```

Arguments

data	The data of analysis
formula	A formula specifying the long-run model equation. This formula defines the relationships between the dependent variable and explanatory variables, including options for deterministic terms, asymmetric variables, and a trend component.

```
Example formula:
\code{y ~ x + z + Asymmetric(z) + Lasymmetric(x2 + x3) + Sasymmetric(x3 + x4) + determi
```

Details

The formula allows flexible specification of variables and their roles:

- **Deterministic variables:** Deterministic regressors (e.g., dummy variables) can be included using `deterministic()`. Multiple deterministic variables may be supplied using `+`, for example `deterministic(dummy1 + dummy2)`. These variables are treated as fixed components and are not associated with short-run or long-run dynamics.

- **Asymmetric variables:** Asymmetric decompositions can be specified for short-run and/or long-run dynamics:
  - **Sasymmetric**: Specifies short-run asymmetric variables. For example, `Sasymmetric(x1 + x2)` applies short-run asymmetric decomposition to `x1` and `x2`.
  - **Lasymmetric**: Specifies long-run asymmetric variables. For example, `Lasymmetric(x1 + x3)` applies long-run asymmetric decomposition to `x1` and `x3`.
  - **Asymmetric**: Specifies variables that enter both short-run and long-run asymmetric decompositions. For example, `Asymmetric(x1 + x3)` applies asymmetric decomposition in both dynamics.

A **trend** term may be included to capture deterministic linear time trends by simply adding `trend` to the formula.

The formula also supports the use of `.` to represent all available regressors in the supplied data (excluding the dependent variable), following standard R formula conventions.

All of the operators `Deterministic()`, `Sasymmetric()`, `Lasymmetric()`, and `Asymmetric()` follow the same usage rules:

- They can be freely combined within a single formula, for example:

```
y ~ . +
  Asymmetric(z) +
  Lasymmetric(x2 + x3) +
  Sasymmetric(x3 + x4) +
  deterministic(dummy1 + dummy2) +
  trend
```

- They must not be nested within one another. Valid usage: `y ~ x + deterministic(dummy) + Asymmetric(z)`. Invalid usage (to be avoided): `y ~ x + deterministic(Asymmetric(z))` or `y ~ x + Asymmetric(deterministic(dummy))`.
- Where applicable, arguments are validated internally using `match.arg()`. Consequently, abbreviated inputs are accepted provided they uniquely identify a valid option. For example, if `"asymmetric"` is an admissible value, specifying `"a"` is sufficient. For clarity and reproducibility, however, full argument names are recommended.

These components may therefore be combined flexibly to construct a specification tailored to the empirical analysis.

`maxlag` An integer specifying the maximum number of lags to be considered for the model. The default value is `4`. This parameter sets an upper limit on the lag length during the model estimation process.

details

The `maxlag` parameter is crucial for defining the maximum lag length that the model will evaluate when selecting the optimal lag structure based on the specified `criterion`. It controls the computational effort and helps prevent overfitting by restricting the search space for lag selection.

- If the data has a short time horizon or is prone to overfitting, consider reducing `maxlag`.

- If the data is expected to have long-term dependencies, increasing `maxlag` may be necessary to capture the relevant dynamics.

Setting an appropriate value for `maxlag` depends on the nature of your dataset and the context of the analysis:

- For small datasets or quick tests, use smaller values (e.g., `maxlag = 2`).
- For datasets with more observations or longer-term patterns, larger values (e.g., `maxlag = 8`) may be appropriate, though this increases computational time.

examples

Using the default maximum lag (4)

```
kardl(data, MyFormula, maxlag = 4)
```

Reducing the maximum lag to 2 for faster computation

```
kardl(data, MyFormula, maxlag = 2)
```

Increasing the maximum lag to 8 for datasets with longer dependencies

```
kardl(data, MyFormula, maxlag = 8)
```

`mode` Specifies the mode of estimation and output control. This parameter determines how the function handles lag estimation and what kind of feedback or control is provided during the process. The available options are:

- **"quick"** (default): Displays progress and messages in the console while the function estimates the optimal lag values. This mode is suitable for interactive use or for users who want to monitor the estimation process in real-time. It provides detailed feedback for debugging or observation but may use additional resources due to verbose output.
- **"grid"** : Displays progress and messages in the console while the function estimates the optimal lag values. This mode is suitable for interactive use or for users who want to monitor the estimation process in real-time. It provides detailed feedback for debugging or observation but may use additional resources due to verbose output.
- **"grid_custom"**: Suppresses most or all console output, prioritizing faster execution and reduced resource usage on PCs or servers. This mode is recommended for high-performance scenarios, batch processing, or when the estimation process does not require user monitoring. Suitable for large-scale or repeated runs where output is unnecessary.
- **User-defined vector**: A numeric vector of lag values specified by the user, allowing full customization of the lag structure used in model estimation. When a user-defined vector is provided (e.g., `c(1, 2, 4, 5)`), the function skips the lag optimization process and directly uses the specified lags.
  Users can define lag values directly as a numeric vector. For example: `mode = c(1, 2, 4, 5)` assigns lags of 1, 2, 4, and 5 to variables in the specified order. Alternatively, lag values can be assigned to variables by name for clarity and control. For example: `mode = c(CPI = 2, ER_POS = 3, ER_NEG = 1, PPI = 3)` assigns lags to variables explicitly. Ensure that the lags are correctly designated by verifying the result using `kardl_model$properLag` after estimation.
  Attention! A function-based criterion or user-defined function can be specified for model selection, but this is only supported for `mode = "grid_custom"`

and `mode = "quick"`. The `mode = "grid"` option is restricted to predefined criteria (e.g., AIC or BIC). For more information on available criteria, see the `modelCriterion` function documentation.

- When using a numeric vector, ensure the order of lag values matches the variables in your formula.
- If using named vectors, double-check the variable names to avoid mismatches or unintended results.
- This mode bypasses the automatic lag optimization and assumes the user-defined lags are correct.

`criterion` A string specifying the information criterion to be used for selecting the optimal lag structure. The available options are:

- **"AIC"**: Akaike Information Criterion (default). This criterion balances model fit and complexity, favoring models that explain the data well with fewer parameters.
- **"BIC"**: Bayesian Information Criterion. This criterion imposes a stronger penalty for model complexity than AIC, making it more conservative in selecting models with fewer parameters.
- **"AICc"**: Corrected Akaike Information Criterion. This is an adjusted version of AIC that accounts for small sample sizes, making it more suitable when the number of observations is limited relative to the number of parameters.
- **"HQ"**: Hannan-Quinn Information Criterion. This criterion provides a compromise between AIC and BIC, favoring models that balance fit and complexity without being overly conservative.

The criterion can be specified as a string (e.g., `"AIC"`) or as a user-defined function that takes a fitted model object. Please visit the `modelCriterion` function documentation for more details on using custom criteria.

`differentAsymLag` A logical value indicating whether to allow different lag lengths for positive and negative decompositions.

`batch` A string specifying the batch processing configuration in the format "current_batch/total_batches". If a user utilize grid or grid_custom mode and want to split the lag search into multiple batches, this parameter can be used to define the current batch and the total number of batches. For example, "2/5" indicates that the current batch is the second out of a total of five batches. The default value is "1/1", meaning that the entire lag search is performed in a single batch.

`...` Additional arguments that can be passed to the function. These arguments can be used to specify other settings or parameters that are not explicitly defined in the main arguments.

Details

The general formula for the long-run model is specified as follows:

$$\Delta y_t = c + \eta_0 y_{t-1} + \sum_{j=1}^{p} \gamma_j \Delta y_{t-j} + \sum_{i=1}^{m} (\eta_i^+ x_{i,t-1}^+ + \eta_i^- x_{i,t-1}^-) + \sum_{i=m+1}^{k} \eta_i x_{i,t-1} + \sum_{i=1}^{m} \sum_{j=0}^{q_i^+} \beta_{ij}^+ \Delta x_{i,t-j}^+ + \sum_{i=1}^{m} \sum_{j=0}^{q_i^-} \beta_{ij}^- \Delta x$$

Where:

- y_t is the dependent variable at time t.
- c is the constant term.
- η_0 is the coefficient of the lagged dependent variable.
- γ_j are the coefficients of the lagged differences of the dependent variable.
- η_i^+ and η_i^- are the coefficients of the positive and negative decompositions of the independent variables, respectively.
- η_i are the coefficients of the independent variables that do not have asymmetric decompositions.
- β_{ij}^+ and β_{ij}^- are the coefficients of the lagged differences of the positive and negative decompositions of the independent variables, respectively.
- β_{ij} are the coefficients of the lagged differences of the independent variables that do not have asymmetric decompositions.
- e_t is the error term at time t.
- p is the maximum lag length for the dependent variable.
- q_i^+ and q_i^- are the maximum lag lengths for the positive and negative decompositions of the independent variables, respectively.
- q_i is the maximum lag length for the independent variables that do not have asymmetric decompositions.
- m is the number of independent variables with asymmetric decompositions.
- k is the total number of independent variables.
- Δ denotes the first difference operator.
- $x_{i,t}^+$ and $x_{i,t}^-$ represent the positive and negative decompositions of the independent variable x_i at time t, respectively.

Value

An object of class `kardl_lm` containing the estimated ARDL or NARDL model. The object includes the following components:

argsInfo A list of input arguments used for the estimation. It includes the data, formula, maxlag, mode, criterion, differentAsymLag, and batch settings.

extractedInfo A list containing extracted information from the input data and formula, such as variable names, deterministic terms, asymmetric variables, and the prepared dataset for estimation.

timeInfo A list containing timing information for the estimation process, including start time, end time, and total duration.

lagInfo A list containing lag selection information, including the optimal lag configuration and criteria values for different lag combinations.

estInfo A list containing estimation details, such as the type of model, estimation method, model formula, number of parameters (k), number of observations (n), start and end points of the fitted values, and total time span.

model The fitted linear model object of class `lm` representing the estimated ARDL or NARDL model.

Notation of reported coefficients

In the reported coefficients, the prefix `L` denotes lagged variables, where the accompanying number indicates the lag order, and `.d.` denotes first differences. Accordingly, `L1.PPI` represents the first lag of the level of `PPI` (long-run component), while `L3.d.PPI` denotes the third lag of the first-differenced `PPI` (short-run component).

In addition, the suffixes `_POS` and `_NEG` indicate the positive and negative partial sum components of a variable, respectively. This notation is used by default and remains valid unless modified through the `kardl_set()` function.

See Also

ecm, kardl_set, kardl_get, kardl_reset, modelCriterion

Examples

```
# Sample article: THE DYNAMICS OF EXCHANGE RATE PASS-THROUGH TO DOMESTIC PRICES IN TURKEY

kardl_set(formula =CPI~ER+PPI+Asymmetr(ER)+deterministic(covid)+trend ,
          data=imf_example_data,
          maxlag=2
) # setting the default values of the kardl function


# using the grid_custom mode with batch processing

kardl_model_grid<-kardl( mode = "grid_custom",batch = "2/3",criterion = "BIC")
kardl_model_grid

kardl_model2<-kardl(mode = c( 2    ,  1    ,  1   ,   3 ))

# Getting the results
kardl_model2

# Getting the summary of the results
summary(kardl_model2)

# using '.' in the formula means that all variables in the data will be used

fit_bic <- kardl(formula=CPI~.+deterministic(covid))
fit_bic

# Setting max lag instead of default value [4]
kardl(imf_example_data,
      CPI~ER+PPI+Lasymmetric(ER),
      maxlag = 3, mode = "grid_custom")

# Using another criterion for finding the best lag
kardl_set(criterion = "HQ") # setting the criterion to HQ
kardl( mode = "grid_custom")
```

```
# using default values of lags
kardl( mode=c(1,2,3,0))

# For using different lag values for negative and positive decompositions of non-linear variables
# setting the same lags for positive and negative decompositions.

same<-kardl(formula=CPI~Asymmetric(ER),maxlag=2, mode = "grid_custom",differentAsymLag = FALSE)
dif<-kardl(formula=CPI~Sasymmetric(ER),maxlag=2, mode = "grid_custom",differentAsymLag = TRUE)

same$lagInfo$OptLag
dif$lagInfo$OptLag

# Optional: use magrittr if available

library(magrittr)
  kardl_model_pipe <-  imf_example_data %>%
    kardl(mode = "grid_custom")

  kardl_model_pipe
```

---

kardl_get *Get kardl Package Options*

Description

This function retrieves the current settings of the kardl package. Users can specify option names to get their values or call the function without arguments to retrieve all current settings.

Usage

```
kardl_get(...)
```

Arguments

...	Option names to retrieve. If no arguments are provided, all options will be returned.

Value

If no arguments are provided, returns all options as a list. If specific option names are provided, returns their values.

See Also

kardl_set, kardl_reset

Examples

```
# Get all options
kardl_get()
# Get specific options
kardl_get("maxlag", "mode")

# Note: In interactive use, avoid calling kardl_get() directly to prevent cluttering the console.

a<-kardl_get()
a$AsymSuffix

# To reset options and then get a default option:
mydefaults <- kardl_reset()
mydefaults$maxlag
```

`kardl_longrun` *Compute Long-Run Multipliers from a kardl Model*

Description

This function calculates the long-run parameters of a KARDL model estimated using the `kardl` function. The long-run parameters are calculated by dividing the negative of the coefficients of the independent variables by the coefficient of the dependent variable. If an intercept is included in the model, it is also standardized by dividing it by the negative of the long-run parameter of the dependent variable.

Usage

```
kardl_longrun(model)
```

Arguments

`model`	An object of class `kardl` estimated using the `kardl` function.

Details

The function also calculates the standard errors of the long-run multipliers using the delta method, which accounts for the covariance between the coefficients. The fitted values and residuals of the long-run model are calculated based on the original data and the long-run multipliers.

The function returns an object of class `kardl_long_run`, which contains the long-run multipliers, their standard errors, t-statistics, p-values, fitted values, residuals, and other relevant information for further analysis and diagnostics.

Note that the fitted values and residuals from the long-run model are not centered (i.e., they do not have a mean of zero) by design, which means that diagnostic plots and residual-based tests may not be valid for this model. The primary focus of this function is on the estimation of the long-run multipliers and their associated statistics.

The long-run multipliers are calculated using the formula:

$$LRM_i = -\frac{\eta_i}{\eta_0}$$

.

t-values and p-values are calculated using the standard errors obtained from the delta method, which accounts for the covariance between the coefficients. Delta method formula for standard errors of long-run multipliers:

$$SE(LR_i) = \sqrt{(A^2) \cdot Var(\eta_i) + 2 \cdot A \cdot B \cdot Cov(\eta_i, \eta_0) + (B^2) \cdot Var(\eta_0)}$$

where

$$A = \frac{\partial LRM_i}{\partial \eta_i} = -\frac{1}{\eta_0}$$

and

$$B = \frac{\partial LRM_i}{\partial \eta_0} = \frac{\eta_i}{\eta_0^2}$$

. Hence, η_i is the coefficient of the independent variable and η_0 is the coefficient of the dependent variable in the original KARDL model.

Value

An object of class `kardl_long_run`, which is a list containing:

- `coefficients`: A named vector of long-run multipliers.
- `residuals`: A vector of residuals from the long-run model.
- `effects`: A vector of effects from the long-run model.
- `rank`: The rank of the long-run model.
- `fitted.values`: A vector of fitted values from the long-run model.
- `assign`: A vector indicating the assignment of coefficients to terms in the long-run model.
- `qr`: The QR decomposition of the design matrix of the long-run model.
- `df.residual`: The degrees of freedom of the residuals of the long-run model.
- `xlevels`: A list of factor levels used in the long-run model.
- `call`: The matched call used to create the long-run model.
- `terms`: The terms object of the long-run model.
- `model`: The data frame used in the long-run model.

See Also

`kardl`, `pssf`, `psst`

Examples

```
kardl_model<-kardl(imf_example_data,
                   CPI~ER+PPI+asym(ER)+deterministic(covid)+trend,
                   mode=c(1,2,3,0))
long<-kardl_longrun(kardl_model)

# Calculate the long-run multipliers
long
# Details of the long-run multipliers
summary(long)


# Using magrittr


library(magrittr)
     imf_example_data %>%
     kardl(CPI~ER+PPI+asym(ER)+deterministic(covid)+trend, mode=c(1,2,3,0)) %>%
     kardl_longrun() %>% summary()
```

`kardl_reset` *Reset kardl Package Options to Default Values*

Description

This function resets kardl package options to their default values.

Usage

```
kardl_reset(exclude = NULL)
```

Arguments

`exclude`	Character vector of setting names that should not be reset. These settings retain their current values. By default, all settings are reset.

Value

A list of the settings after reset, returned invisibly.

See Also

`kardl_set`, `kardl_get`

Examples

```
kardl_set(criterion = "BIC", differentAsymLag = FALSE)

# Reset all settings to defaults except "criterion"
kardl_reset(exclude = "criterion")

# Get the current settings to verify the reset
print(kardl_get("criterion"))

# This will show "BIC" since it was excluded from the reset,
#while other settings will be reset to their defaults.
print(kardl_get("differentAsymLag"))
```

`kardl_set` *Set kardl Package Options*

Description

This function allows users to set options for the kardl package. Users can specify named arguments to set options or call the function without arguments to retrieve all current settings.

Usage

```
kardl_set(...)
```

Arguments

`...`	Named arguments corresponding to the options to be set. Valid option names include those defined in the kardl package settings.

Value

All current settings as a list after applying any updates from the provided named arguments invisibly.

If no arguments are provided, returns all options as a list. If named arguments are provided, sets those options and returns the updated list.

See Also

`kardl_get`, `kardl_reset`

Examples

```
# Get default options
kardl_get("maxlag", "mode")
# Set options
kardl_set(maxlag = 5, mode = "grid")
# Get specific options
kardl_get("maxlag", "mode")
```

```
# To have the updated settings available in the global environment, assign the output to a variable:
MySettings<- kardl_set(LongCoef = "LongRun_{varName}", ShortCoef = "ShortRun_{varName}")
# Now MySettings contains the updated settings, and the kardl package
# will use these settings for subsequent operations.
MySettings$LongCoef
MySettings$maxlag

# Reset to defaults after demonstrating custom settings
kardl_reset()
```

`lmerge` *Merge Lists with Priority to the First Argument*

Description

The first list of values takes precedence. When both lists have items with the same names, the values from the first list will be applied. In merging the two lists, priority is given to the left list, so if there are overlapping items, the corresponding value from the left list will be used in the merged result.

Usage

```
lmerge(first, second, ...)
```

Arguments

`first`	The first list
`second`	The second list
`...`	Additional lists to merge

Details

The `lmerge()` function is designed to merge multiple lists while giving precedence to the values in the first list. This function is particularly useful when you want to combine settings or parameters from multiple sources while ensuring that the primary source (the first list) takes priority over others.

For right merge, a user can simply swap the order of the lists to give priority to the second list. For example, `lmerge(b, a)` will prioritize values from list `b` over those in list `a`.

Value

A merged list with unique names, prioritizing values from the first list in case of name conflicts.

See Also

append

Examples

```
a<-list("a"="first a","b"="second a","c"=list("w"=12,"k"=c(1,3,6)))
b<-list("a"="first b","b"="second b","d"=14,"e"=45)
myMerged<- lmerge(a,b)
print(unlist(myMerged))

# for right merge
myMerged<- lmerge(b,a)
print(unlist(myMerged))

# for more than two lists
myMerged<- lmerge(a,b,c("v1"=11,22,3,"v5"=5))
print(unlist(myMerged))

# for more than two lists with nested lists
m2<-list("m1"="kk2","m1.2.3"=list("m1.1.1"=333,"m.1.4"=918,"m.1.5"=982,"m.1.6"=981,"m.1.7"=928))
m3<-list("m1"="kk23","m2.3"=2233,"m1.2.4"=list("m1.1.1"=333444,"m.1.5"=982,"m.1.6"=91,"m.1.7"=928))
a<-c(32,34,542,"k"=35)
b<-c(65,"k"=34)

h1<-lmerge(a, m2)
print(unlist(h1))

h2<-lmerge(a,b,m2,m3,list("m1.1"=4))
print(unlist(h2))
```

---

modelCriterion *Model Selection Criteria*

Description

Computes a model selection criterion (AIC, BIC, AICc, or HQ) or applies a user-defined function to evaluate a statistical model.

Usage

```
modelCriterion(estModel, cr, ...)
```

Arguments

estModel	An object containing the fitted model. The object should include at least: • estModel$model - the actual fitted model object (e.g., from lm, glm). • k - the number of estimated parameters. • n - the sample size.
cr	A character string specifying the criterion to compute. Options are "AIC", "BIC", "AICc", and "HQ". Alternatively, a user-defined function can be provided. See details below for more information on using custom criteria.

... Additional arguments passed to the user-defined criterion function if `cr` is a function.

Details

This function returns model selection criteria used to compare the quality of different models. All criteria are defined such that **lower values indicate better models** (i.e., the goal is minimization).

If you wish to compare models using a maximization approach (e.g., log-likelihood), you can multiply the result by `-1`.

Note: The predefined string options (e.g., `"AIC"`) are **not** the same as the built-in R functions `AIC()` or `BIC()`. In particular, the values returned by this function are adjusted by dividing by the sample size `n` (i.e., normalized AIC/BIC), which makes it more comparable across datasets of different sizes.

The function returns:

- **"AIC"**: $\frac{2k-2\ell}{n}$ Akaike Information Criterion divided by `n`.
- **"BIC"**: $\frac{\log(n)\cdot k-2\ell}{n}$ Bayesian Information Criterion divided by `n`.
- **"AICc"**: $\frac{2k(k+1)}{n-k-1} + \frac{2k-2\ell}{n}$ Corrected Akaike Information Criterion divided by `n`.
- **"HQ"**: $\frac{2\log(\log(n))\cdot k-2\ell}{n}$ Hannan-Quinn Criterion divided by `n`.

where:

- k is the number of parameters,
- n is the sample size,
- ℓ is the log-likelihood of the model.

If `cr` is a function, it is called with the fitted model and any additional arguments passed through `....`

Value

A numeric value representing the selected criterion, normalized by the sample size if one of the predefined options is used.

See Also

`kardl`

Examples

```
# Example usage of modelCriterion function with a simple linear model
mylm<- lm(mpg ~ wt + hp, data = mtcars)
modelCriterion(mylm, AIC )
modelCriterion(mylm, "AIC" )

 # Example usage of modelCriterion function with a kardl model
 kardl_model <- kardl(imf_example_data,
                      CPI ~ ER + PPI + asym(ER) + deterministic(covid) + trend,
```

```
                    mode = c(1, 2, 3, 0))

# Using AIC as the kardl package's built-in criterion function which is different
# from the base R AIC function.
modelCriterion(kardl_model, "AIC")

# Using the base R AIC function directly on the fitted model object
modelCriterion(kardl_model, AIC)
# Using the base R AIC function outside of modelCriterion to compute AIC for the fitted model
AIC(kardl_model)

# Using BIC as the criterion for the kardl model which is different from the base R BIC function.
modelCriterion(kardl_model, "BIC")

# Using a custom criterion function that divides AIC by the sample size
my_cr_fun <- function(mod, ...) { AIC(mod) / length(mod$model[[1]]) }
modelCriterion(kardl_model, my_cr_fun)
```

`mplier` *Compute Dynamic Multipliers for kardl Models*

Description

Computes cumulative dynamic multipliers based on a model estimated using the `kardl` framework. The function supports different configurations of linearity and asymmetry in both short-run and long-run dynamics.

Usage

```
mplier(kmodel, horizon = 80, minProb = 0)
```

Arguments

`kmodel`	An object of class `kardl_lm` produced by the `kardl` function.
`horizon`	Integer. Number of periods ahead for which dynamic multipliers are computed.
`minProb`	Numeric. Minimum p-value threshold for including coefficients in the calculation. Coefficients with p-values above this threshold will be set to zero. Default is `0` (no threshold). This parameter allows users to control the inclusion of coefficients in the calculation based on their statistical significance. Setting a threshold can help focus the analysis on more relevant variables, but it may also exclude potentially important effects if set too stringently.

Details

The asymmetry structure is determined internally:

- Variables in `extractedInfo$ASvars` are treated as asymmetric in the short run.

- Variables in `extractedInfo$ALvars` are treated as asymmetric in the long run.

This allows four possible configurations:

- **LL**: Linear in both short-run and long-run
- **NN**: Asymmetric in both short-run and long-run
- **SA**: Short-run linear, long-run asymmetric
- **AS**: Short-run asymmetric, long-run linear

When a component is linear, the same coefficient path is used for both positive and negative changes. When asymmetric, separate positive and negative effects are computed.

The `mplier` function computes dynamic multipliers based on the coefficients and lag structure of a model estimated using the `kardl` package. The function extracts necessary information from the model, such as coefficients, lag structure, and variable names, to compute the dynamic multipliers. It calculates the short-run coefficients, Lambda values, and omega values based on the model's parameters and lag structure. The output includes a matrix of dynamic multipliers (mpsi), which can be used for further analysis or visualization. The dynamic multipliers provide insight into how changes in the independent variables affect the dependent variable over time, allowing for a deeper understanding of the relationships captured by the model. The function also allows users to set a minimum p-value threshold for including coefficients in the calculation, providing flexibility in focusing on statistically significant effects.

The function constructs dynamic multipliers based on the recursive relationship:

$$\psi_h^+ = \sum_{i=0}^{h} \frac{\partial y_{t+i}}{\partial x_t^+}, \quad \psi_h^- = \sum_{i=0}^{h} \frac{\partial y_{t+i}}{\partial x_t^-}$$

where ψ_h^+ and ψ_h^- represent cumulative responses to positive and negative shocks.

The recursion is defined as:

$$\psi_h = \lambda_h + \sum_{j=1}^{p} \omega_j \psi_{h-j}$$

where λ_h captures short-run effects and ω_j reflects persistence through lagged dependent variables.

When asymmetry is present, positive and negative shocks are propagated separately. Otherwise, the same dynamic path is used.

Value

A list of class `kardl_mplier` containing:

- **mpsi**: Matrix of cumulative dynamic multipliers.
- **omega**: Vector of omega coefficients (persistence structure).
- **lambda**: Matrix of short-run dynamic coefficients.
- **horizon**: Forecast horizon used.
- **vars**: Extracted model variable structure.

See Also

bootstrap

Examples

```
# This example demonstrates how to use the mplier function to calculate dynamic multipliers
# from a model estimated using the kardl package. The example includes fitting a model with
# the kardl function, calculating the multipliers, and visualizing the results using both
# base R plotting and ggplot2.

 # Calculating dynamic multipliers for a linear model in short and long run (NN)

 kardl_model<-kardl(imf_example_data, CPI~ER )
 m<-mplier(kardl_model,40)
 head(m$mpsi)
 plot(m)

 # Calculating dynamic multipliers for a model with
 # Short-run linear, long-run asymmetric (SA)
 kardl_model<-kardl(imf_example_data, CPI~lasym(ER) )
 m<-mplier(kardl_model,40)
 head(m$mpsi)
 plot(m)

 # Calculating dynamic multipliers for a model with
 # Short-run asymmetric, long-run linear (AS)
 kardl_model<-kardl(imf_example_data, CPI~sasym(ER) )
 m<-mplier(kardl_model,40)
 head(m$mpsi)
 plot(m)

 # Calculating dynamic multipliers for a model with
 # asymmetric effects in both short and long run (NN)
 kardl_model<-kardl(imf_example_data, CPI~asym(ER) )
 m<-mplier(kardl_model,40)
 head(m$mpsi)
 plot(m)

# The mpsi matrix contains the cumulative dynamic multipliers for each variable and time horizon.
 # The omega vector contains the persistence structure of the model,
 # while the lambda matrix contains the short-run dynamic coefficients.
 # You can inspect these components to understand the dynamics captured by the model.
 kardl_model<-kardl(imf_example_data, CPI~PPI+asym(ER) )
 m<-mplier(kardl_model,40)
 head(m$mpsi)
 head(m$omega)
 head(m$lambda)

 # For plotting specific variables, you can specify them in the plot function. For example,
 # to plot the multipliers for the variable "PPI":

 plot(m, variable = "PPI")
```

narayan *Narayan Bounds Test*

Description

This function performs the Narayan test, which is designed to assess cointegration using critical values specifically tailored for small sample sizes. Unlike traditional cointegration tests that may rely on asymptotic distributions, the Narayan test adjusts for the limitations of small samples, providing more accurate results in such contexts. This makes the test particularly useful for studies with fewer observations, as it accounts for sample size constraints when determining the presence of a long-term equilibrium relationship between variables.

Usage

```
narayan(kmodel, case = 3, signif_level = "auto")
```

Arguments

kmodel	The kardl obejct
case	Numeric or character. Specifies the case of the test to be used in the function. Acceptable values are 1, 2, 3, 4, 5, and "auto". If "auto" is chosen, the function determines the case automatically based on the model's characteristics. Invalid values will result in an error. • 1: No intercept and no trend. This case is not supported by the Narayan test. • 2: Restricted intercept and no trend. • 3: Unrestricted intercept and no trend. • 4: Unrestricted intercept and restricted trend. • 5: Unrestricted intercept and unrestricted trend.
signif_level	Character or numeric. Specifies the significance level to be used in the function. Acceptable values are "auto", "0.10", "0.1", "0.05", "0.025", and "0.01". If a numeric value is provided, it will be converted to a character string. When "auto" is selected, the function determines the significance level sequentially, starting from the most stringent level ("0.01") and proceeding to "0.025", "0.05", and "0.10" until a suitable level is identified. Invalid values will result in an error.

Value

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

- statistic: The calculated F-statistic for the test.

- `caseTxt`: A character string describing the case used for the test, based on the specified case parameter.
- `alternative`: A character string describing the alternative hypothesis of the test.
- `sample.size`: The number of observations used in the test.
- `varnames`: A character vector containing the names of the dependent variable and independent variables used in the test.
- `k`: The number of independent variables (excluding the dependent variable) included in the test.
- `sig`: The significance level used for the test, either specified by the user or determined automatically.
- `notes`: A character vector containing any notes or warnings related to the test, such as the suitability of the test for small sample sizes or any adjustments made to the case based on the model's characteristics.

Hypothesis testing

The null hypothesis (H0) of the F Bound test is that there is no cointegration among the variables in the model. In other words, it tests whether the long-term relationship between the variables is statistically significant. If the calculated F-statistic exceeds the upper critical value, we reject the null hypothesis and conclude that there is cointegration. Conversely, if the F-statistic falls below the lower critical value, we fail to reject the null hypothesis, indicating no evidence of cointegration. If the F-statistic lies between the two critical values, the result is inconclusive.

$$\Delta y_t = \psi + \varphi t + \eta_0 y_{t-1} + \sum_{i=1}^{k} \eta_i x_{i,t-1} + \sum_{j=1}^{p} \gamma_j \Delta y_{t-j} + \sum_{i=1}^{k} \sum_{j=0}^{q_i} \beta_{ij} \Delta x_{i,t-j} + e_t$$

Cases 1, 3, 5:

$$\mathbf{H_0}: \eta_0 = \eta_1 = \cdots = \eta_k = 0$$

$$\mathbf{H_1}: \eta_0 \neq \eta_1 \neq \ldots \neq \eta_k \neq 0$$

Case 2:

$$\mathbf{H_0}: \eta_0 = \eta_1 = \cdots = \eta_k = \psi = 0$$

$$\mathbf{H_1}: \eta_0 \neq \eta_1 \neq \ldots \neq \eta_k \neq \psi \neq 0$$

Case 4:

$$\mathbf{H_0}: \eta_0 = \eta_1 = \cdots = \eta_k = \varphi = 0$$

$$\mathbf{H_1}: \eta_0 \neq \eta_1 \neq \ldots \neq \eta_k \neq \varphi \neq 0$$

References

Narayan, P. K. (2005). The saving and investment nexus for China: evidence from cointegration tests. Applied economics, 37(17), 1979-1990.

See Also

`pssf` `psst` `ecm`

Examples

```
kardl_model<-kardl(imf_example_data,
                   CPI~ER+PPI+asym(ER)+deterministic(covid)+trend,
                   mode=c(1,2,3,0))
my_test<-narayan(kardl_model)
# Getting the results of the test.
my_test
# Getting details of the test.
my_summary<-summary(my_test)
my_summary

# Getting the critical values of the test.
my_summary$crit_vals




# Using magrittr :


library(magrittr)
  imf_example_data %>%
  kardl(CPI~ER+PPI+asym(ER)+deterministic(covid)+trend,
                        mode=c(1,2,3,0)) %>% narayan()

  # Getting details of the test results using magrittr:
   imf_example_data %>%
   kardl(CPI~ER+PPI+asym(ER)+deterministic(covid)+trend,
                           mode=c(1,2,3,0)) %>%
   narayan() %>% summary()
```

`parse_formula_vars` *Parse Formula Variables*

Description

The `parseFormula()` function analyzes a given formula to identify and extract variables that match specified patterns. It is particularly useful for isolating variables enclosed within certain functions or constructs in the formula, such as `asym()`, `det()`, or any user-defined patterns.

Usage

```
parse_formula_vars(formula)
```

Arguments

`formula`	The initial formula for the model, typically specified using R's formula syntax (e.g., `y ~ x + f(x1 + x2)`).

Value

A list containing:

- `response`: The response variable(s) extracted from the formula.
- `intercept`: A logical value indicating whether the formula includes an intercept (default is TRUE).
- `dot`: A logical value indicating whether the formula includes a dot (.) representing all other variables (default is FALSE).
- `outside`: A vector of variables that are outside any specified patterns. It includes the variables that are not detected within the specified patterns in the formula.
- `inside`: A list where each element corresponds to a detected pattern (e.g., function name) and contains the variables found inside that pattern. For example, if the formula includes `asym(x1 + x2)`, the `inside` list will have an element named "asym" containing the variables "x1" and "x2". This allows for easy identification of variables that are part of specific constructs in the formula.

See Also

`formula` and `gregexpr`

Examples

```
# Parse formulas containing various collection types like ()
formula_ <- y ~ x +det(s -gg- d) + asymS(d2 -rr+ s)-mm(y1+y2+y3)+asym(k1+k2+k3)+trend-huseyin
# Extract variables
parse_formula_vars(formula_)
```

---

pssf *Pesaran, Shin, and Smith Bounds F-Test*

Description

This function performs the Pesaran, Shin, and Smith (PSS) F Bound test to assess the presence of a long-term relationship (cointegration) between variables in the context of an autoregressive distributed lag (ARDL) model. The PSS F Bound test examines the joint significance of lagged levels of the variables in the model. It provides critical values for both the upper and lower bounds, which help determine whether the variables are cointegrated. If the calculated F-statistic falls outside these bounds, it indicates the existence of a long-term equilibrium relationship. This test is particularly useful when the underlying data includes a mix of stationary and non-stationary variables.

Usage

```
pssf(kmodel, case = 3, signif_level = "auto")
```

Arguments

kmodel	A fitted KARDL model object of class 'kardl_lm' created using the kardl function.
case	Numeric or character. Specifies the case of the test to be used in the function. Acceptable values are 1, 2, 3, 4, 5, and "auto". If "auto" is chosen, the function determines the case automatically based on the model's characteristics. Invalid values will result in an error. • 1: No intercept and no trend • 2: Restricted intercept and no trend • 3: Unrestricted intercept and no trend • 4: Unrestricted intercept and restricted trend • 5: Unrestricted intercept and unrestricted trend
signif_level	Character or numeric. Specifies the significance level to be used in the function. Acceptable values are "auto", "0.10", "0.1", "0.05", "0.025", and "0.01". If a numeric value is provided, it will be converted to a character string. When "auto" is selected, the function determines the significance level sequentially, starting from the most stringent level ("0.01") and proceeding to "0.025", "0.05", and "0.10" until a suitable level is identified. Invalid values will result in an error.

Value

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

- statistic: The calculated F-statistic for the test.
- caseTxt: A character string describing the case used for the test, based on the specified case parameter.
- alternative: A character string describing the alternative hypothesis of the test.
- sample.size: The number of observations used in the test.
- varnames: A character vector containing the names of the dependent variable and independent variables used in the test.
- k: The number of independent variables (excluding the dependent variable) included in the test.
- sig: The significance level used for the test, either specified by the user or determined automatically.
- notes: A character vector containing any notes or warnings related to the test, such as the suitability of the test for small sample sizes or any adjustments made to the case based on the model's characteristics.

Hypothesis testing

The null hypothesis (H0) of the F Bound test is that there is no cointegration among the variables in the model. In other words, it tests whether the long-term relationship between the variables is statistically significant. If the calculated F-statistic exceeds the upper critical value, we reject the null hypothesis and conclude that there is cointegration. Conversely, if the F-statistic falls below the

lower critical value, we fail to reject the null hypothesis, indicating no evidence of cointegration. If the F-statistic lies between the two critical values, the result is inconclusive.

$$\Delta y_t = \psi + \varphi t + \eta_0 y_{t-1} + \sum_{i=1}^{k} \eta_i x_{i,t-1} + \sum_{j=1}^{p} \gamma_j \Delta y_{t-j} + \sum_{i=1}^{k} \sum_{j=0}^{q_i} \beta_{ij} \Delta x_{i,t-j} + e_t$$

Cases 1, 3, 5:

$$\mathbf{H_0}: \eta_0 = \eta_1 = \cdots = \eta_k = 0$$

$$\mathbf{H_1}: \eta_0 \neq \eta_1 \neq \ldots \neq \eta_k \neq 0$$

Case 2:

$$\mathbf{H_0}: \eta_0 = \eta_1 = \cdots = \eta_k = \psi = 0$$

$$\mathbf{H_1}: \eta_0 \neq \eta_1 \neq \ldots \neq \eta_k \neq \psi \neq 0$$

Case 4:

$$\mathbf{H_0}: \eta_0 = \eta_1 = \cdots = \eta_k = \varphi = 0$$

$$\mathbf{H_1}: \eta_0 \neq \eta_1 \neq \ldots \neq \eta_k \neq \varphi \neq 0$$

References

Pesaran, M. H., Shin, Y. and Smith, R. (2001), "Bounds Testing Approaches to the Analysis of Level Relationship", Journal of Applied Econometrics, 16(3), 289-326.

See Also

psst ecm narayan

Examples

```
kardl_model<-kardl(imf_example_data,
                   CPI~ER+PPI+asym(ER)+deterministic(covid)+trend,
                   mode=c(1,2,3,0))
my_pssF<-pssf(kardl_model)
# Getting the results of the test.
my_pssF
# Getting details of the test.
my_summary<-summary(my_pssF)
my_summary

# Getting the critical values of the test.
my_summary$crit_vals


# Using magrittr :


library(magrittr)
   imf_example_data %>%
   kardl(CPI~ER+PPI+asym(ER)+deterministic(covid)+trend,
                           mode=c(1,2,3,0)) %>% pssf()
```

```
# Getting details of the test results using magrittr:
imf_example_data %>%
kardl(CPI~ER+PPI+asym(ER)+deterministic(covid)+trend,
                        mode=c(1,2,3,0)) %>%
pssf() %>% summary()
```

---

psst *Pesaran, Shin, and Smith t Bounds Test*

Description

This function performs the Pesaran t Bound test

Usage

```
psst(kmodel, case = 3, signif_level = "auto")
```

Arguments

kmodel	A fitted KARDL model object of class 'kardl_lm' created using the kardl function.
case	Numeric or character. Specifies the case of the test to be used in the function. Acceptable values are 1, 2, 3, 4, 5, and "auto". If "auto" is chosen, the function determines the case automatically based on the model's characteristics. Invalid values will result in an error. • 1: No intercept and no trend • 2: Restricted intercept and no trend • 3: Unrestricted intercept and no trend • 4: Unrestricted intercept and restricted trend • 5: Unrestricted intercept and unrestricted trend
signif_level	Character or numeric. Specifies the significance level to be used in the function. Acceptable values are "auto", "0.10", "0.1", "0.05", "0.025", and "0.01". If a numeric value is provided, it will be converted to a character string. When "auto" is selected, the function determines the significance level sequentially, starting from the most stringent level ("0.01") and proceeding to "0.025", "0.05", and "0.10" until a suitable level is identified. Invalid values will result in an error.

Details

This function performs the Pesaran, Shin, and Smith (PSS) t Bound test, which is used to detect the existence of a long-term relationship (cointegration) between variables in an autoregressive distributed lag (ARDL) model. The t Bound test specifically focuses on the significance of the coefficient of the lagged dependent variable, helping to assess whether the variable reverts to its long-term equilibrium after short-term deviations. The test provides critical values for both upper and lower bounds. If the t-statistic falls within the appropriate range, it confirms the presence of cointegration. This test is particularly useful when working with datasets containing both stationary and non-stationary variables.

Value

The function returns an object of class "htest" containing the following components:

statistic The calculated t-statistic for the test.

method A description of the test performed.

data.name The name of the data used in the test.

k The number of independent variables in the model.

notes Any notes or warnings related to the test results, such as sample size considerations or adjustments made to the case based on model characteristics.

sig The significance level used for the test, either specified by the user or determined automatically.

alternative The alternative hypothesis being tested.

case The case used for the test, either specified by the user or determined automatically based on the model's characteristics.

Hypothesis testing

The PSS t Bound test evaluates the null hypothesis that the long-run coefficients of the model are equal to zero against the alternative hypothesis that at least one of them is non-zero. The test is conducted under different cases, depending on the model specification.

$$\Delta y_t = \psi + \varphi t + \eta_0 y_{t-1} + \sum_{i=1}^{k} \eta_i x_{i,t-1} + \sum_{j=1}^{p} \gamma_j \Delta y_{t-j} + \sum_{i=1}^{k} \sum_{j=0}^{q_i} \beta_{ij} \Delta x_{i,t-j} + e_t$$

$$\mathbf{H_0} : \eta_0 = 0$$

$$\mathbf{H_1} : \eta_0 \neq 0$$

References

Pesaran, M. H., Shin, Y. and Smith, R. (2001), "Bounds Testing Approaches to the Analysis of Level Relationship", Journal of Applied Econometrics, 16(3), 289-326.

See Also

pssf ecm narayan

Examples

```
kardl_model<-kardl(imf_example_data,
                   CPI~ER+PPI+asym(ER)+deterministic(covid)+trend,
                   mode=c(1,2,3,0))
my_test<-psst(kardl_model)
# Getting the results of the test.
my_test
# Getting details of the test.
my_summary<-summary(my_test)
my_summary

# Getting the critical values of the test.
my_summary$crit_vals




# Using magrittr :


library(magrittr)
     imf_example_data %>%
     kardl(CPI~ER+PPI+asym(ER)+deterministic(covid)+trend,
                           mode=c(1,2,3,0))    %>%
                           psst()

   # Getting details of the test results using magrittr:
   imf_example_data %>%
   kardl(CPI~ER+PPI+asym(ER)+deterministic(covid)+trend,
                           mode=c(1,2,3,0)) %>%
   psst() %>% summary()
```

`symmetrytest` *Symmetry Test for Nonlinear kardl Models*

Description

The symmetry test is a statistical procedure used to assess the presence of symmetry in the relationship between variables in a model. It is particularly useful in econometric analysis, where it helps to identify whether the effects of changes in one variable on another are symmetric or asymmetric. The test involves estimating a model that includes both positive and negative components of the variables and then performing a Wald test to determine if the coefficients of these components are significantly different from each other. If the test indicates significant differences, it suggests that the relationship is asymmetric, meaning that the impact of increases and decreases in the variables differs. This test returns results for both long-run and short-run variables in a KARDL model. Where applicable, it provides the Wald test statistics, p-values, degrees of freedom, sum of squares,

and mean squares for each variable tested. If the null hypothesis of symmetry is rejected, it indicates that the effects of positive and negative changes in the variable are significantly different, suggesting an asymmetric relationship.

The non-linear model with one asymmetric variables is specified as follows:

$$\Delta y_t = \psi + \eta_0 y_{t-1} + \eta_1^+ x_{t-1}^+ + \eta_1^- x_{t-1}^- + \sum_{j=1}^{p} \gamma_j \Delta y_{t-j} + \sum_{j=0}^{q} \beta_j^+ \Delta x_{t-j}^+ + \sum_{j=0}^{m} \beta_j^- \Delta x_{t-j}^- + e_t$$

This function performs the symmetry test both for long-run and short-run variables in a kardl model. It uses the `nlWaldtest` function from the **nlWaldTest** package for long-run variables and the `linearHypothesis` function from the **car** package for short-run variables. The hypotheses for the long-run variables are:

$$H_0 : -\frac{\eta_1^+}{\eta_0} = -\frac{\eta_1^-}{\eta_0}$$

$$H_1 : -\frac{\eta_1^+}{\eta_0} \neq -\frac{\eta_1^-}{\eta_0}$$

The hypotheses for the short-run variables are:

$$H_0 : \sum_{j=0}^{q} \beta_j^+ = \sum_{j=0}^{m} \beta_j^-$$

$$H_1 : \sum_{j=0}^{q} \beta_j^+ \neq \sum_{j=0}^{m} \beta_j^-$$

Usage

```
symmetrytest(kmodel, vars = NULL, component = "both", type = "F", ...)
```

Arguments

`kmodel`	The kardl obejct
`vars`	A character vector specifying the names of the variables to be included in the symmetry test. If NULL (the default), all eligible variables will be tested. The variable names should match those used in the KARDL model, and they can be either long-run or short-run variables. If any specified variable is not found in the model, an error will be raised.
`component`	A character string specifying which component(s) of the model to test for symmetry. The options are "both" (default), "shortrun", or "longrun". If "both" is selected, the function will perform symmetry tests for both long-run and short-run variables. If "shortrun" is selected, only short-run variables will be tested, and if "longrun" is selected, only long-run variables will be tested. Invalid values will result in an error.
`type`	A character string specifying the type of test statistic to be used in the Wald tests. The options are "F" (default) or "Chisq". If "F" is selected, the function will perform an F-test, which is appropriate for smaller sample sizes. If "Chisq" is selected, the function will perform a chi-squared test, which is more suitable for larger sample sizes. Invalid values will result in an error.

... Additional arguments to be passed to the underlying test functions, such as nlWaldtest for long-run tests or linearHypothesis for short-run tests. These arguments can include options for controlling the behavior of the tests, such as specifying the type of test statistic or adjusting for multiple comparisons.

Details

This function performs symmetry tests on non-linear KARDL models to assess whether the effects of positive and negative changes in independent variables are statistically different.

This function evaluates whether the inclusion of a particular variable in the model follows a linear relationship or exhibits a non-linear pattern. By analyzing the behavior of the variable, the function helps to identify if the relationship between the variable and the outcome of interest adheres to a straight-line assumption or if it deviates, indicating a non-linear interaction. This distinction is important in model specification, as it ensures that the variable is appropriately represented, which can enhance the model's accuracy and predictive performance.

Value

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

- `Lwald`: A data frame containing the Wald test results for the long-run variables, including F-statistic, p-value, degrees of freedom, and residual degrees of freedom.
- `Lhypotheses`: A list containing the null and alternative hypotheses for the long-run variables.
- `Swald`: A data frame containing the Wald test results for the short-run variables, including F-statistic, p-value, degrees of freedom, residual degrees of freedom, and sum of squares.
- `Shypotheses`: A list containing the null and alternative hypotheses for the short-run variables.

References

Shin, Y., Yu, B., & Greenwood-Nimmo, M. (2014). Modelling asymmetric cointegration and dynamic multipliers in a nonlinear ARDL framework. Festschrift in honor of Peter Schmidt: Econometric methods and applications, 281-314.

See Also

kardl, pssf, psst, ecm, narayan

Examples

```
kardl_model<-kardl(imf_example_data,
                   CPI~Lasym(PPI+ER)+Sas(ER)+deterministic(covid)+trend)
ast<- symmetrytest(kardl_model)
ast
# Detailed results of the test:
summary(ast)
# The null hypothesis of the test is that the model is symmetric, while the alternative
# hypothesis is that the model is asymmetric. The test statistic and p-value are provided
# in the output. If the p-value is less than a chosen significance level (e.g., 0.05),
# we reject the null hypothesis and conclude that there is evidence of asymmetry in the model.
```

```
# The default significance level is 0.05, but you can specify a different level using the 'level'
# argument in the summary function. For example, to use a significance level of 0.01,
# you can use the following code:
summary(ast, level = 0.01)

# To get symmetry test results in long-run, you can use the following code:
ast$Lwald

# To get symmetry test results in short-run, you can use the following code:
ast$Swald

# To get the null and alternative hypotheses of the test in long-run,
# you can use the following code:

ast$Lhypotheses

# To get the null and alternative hypotheses of the test in short-run,
# you can use the following code:

ast$Shypotheses
# Alternatively, you can also use the symmetrytest function with the component
# argument to specify whether you want to test for long-run or short-run symmetry.
# For example, to test for long-run symmetry, you can use the following code:
symmetrytest(kardl_model, component = "longrun")
# To test for short-run symmetry, you can use the following code:
symmetrytest(kardl_model, component = "shortrun")

# If you want to test for symmetry with respect to a specific variable,
# you can use the vars argument in the symmetrytest function. For example,
# to test for symmetry with respect to the PPI variable, you can use the following code:
symmetrytest(kardl_model, vars = "PPI")

# To test for symmetry with respect to multiple variables, you can provide
# a vector of variable names to the vars argument. For example, to test for
# symmetry with respect to both PPI and ER, you can use the following code:
symmetrytest(kardl_model, vars = c("PPI", "ER"))

# Finally, you can also specify the type of test statistic to be used in the
# symmetry test. By default, the function uses the Wald test  F statistic,
# but you can also choose to use the chi-squared test statistic.
# For example, to use the chi-squared test statistic, you can use the following code:
symmetrytest(kardl_model, type="Chisq")
```

Index